An analysis of algorithms to estimate the characteristics of the underlying population in Massively Parallel Pyrosequencing data

Anisa Ragalo

A thesis submitted to the University of KwaZulu-Natal,
College of Agriculture, Engineering and Science,
for the degree of
Masters (Msc.) in Computer Science

Supervised by Professor Hugh Murrell

School of Computer Science
University of KwaZulu-Natal
December, 2011

I, **Anisa Ragalo**, declare that:

1. The research reported in this dissertation/thesis, except where otherwise indicated, is my original research.
2. This thesis has not been submitted for any degree or examination at any other university.
3. This thesis does not contain other persons data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This thesis does not contain other persons writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
    - their words have been re-written but the general information attributed to them has been referenced:
    - where their exact words have been used, their writing has been placed inside quotation marks, and referenced.
5. This dissertation/thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation/ thesis and in the References sections.

Candidate: *Anisa Ragalo*

Signature:

As the candidates supervisor I agree to the submission of this thesis for examination.

Supervisor: *Professor Hugh Murrell*

Signature:

Abstract

Massively Parallel Pyrosequencing (MPP) is a next generation DNA sequencing technique that is becoming ubiquitous because it is considerably faster, cheaper and produces a higher throughput than long established sequencing techniques like Sanger sequencing. The MPP methodology is also much less labor intensive than Sanger sequencing.

Indeed, MPP has become a preferred technology in experiments that seek to determine the distinctive genetic variation present in homologous genomic regions.

However there arises a problem in the interpretation of the reads derived from an MPP experiment. Specifically MPP reads are characteristically error prone. This means that it becomes difficult to separate the authentic genomic variation underlying a set of MPP reads from variation that is a consequence of sequencing error.

The difficulty of inferring authentic variation is further compounded by the fact that MPP reads are also characteristically short. As a consequence of this, the correct alignment of an MPP read with respect to the genomic region from which it was derived may not be intuitive.

To this end, several computational algorithms that seek to correctly align and remove the non-authentic genetic variation from MPP reads have been proposed in literature. We refer to the removal of non-authentic variation from a set of MPP reads as error correction. Computational algorithms that process MPP data are classified as sequence-space algorithms and flow-space algorithms. Sequence-space algorithms work with MPP sequencing reads as raw data, whereas flow-space algorithms work with MPP flowgrams as raw data. A flowgram is an intermediate product of MPP, which is subsequently converted into a sequencing read.

In theory, flow-space computations should produce more accurate results than sequence-space computations.

In this thesis, we make a qualitative comparison of the distinct solutions delivered by selected MPP read alignment algorithms. Further we make a qualitative comparison of the distinct solutions delivered by selected MPP error correction algorithms.

Our comparisons between different algorithms with the same niche are facilitated by the design of a platform for MPP simulation, PyroSim. PyroSim is designed to encapsulate the error rate that is characteristic of MPP.

We implement a selection of sequence-space and flow-space alignment algorithms in a software package, MPPAlign. We derive a quality ranking for the distinct algorithms implemented in MPPAlign through a series of qualitative comparisons.

Further, we implement a selection of sequence-space and flow-space error correction algorithms in a software package, MPPErrorCorrect. Similarly, we derive a quality ranking for the distinct algorithms implemented in MPPErrorCorrect through a series of qualitative comparisons.

Contrary to the view expressed in literature which postulates that flow-space computations are more accurate than sequence-space computations, we find that in general the sequence-space algorithms that we implement outperform the flow-space algorithms.

We surmise that flow-space is a more sensitive domain for conducting computations and can only yield consistently good results under stringent quality control measures. In sequence-space, however, we find that base calling, the process that converts flowgrams (flow-space raw data) into sequencing reads (sequence-space raw data), leads to more reliable computations.

To Sheila and Brian, for holding my hand and walking with me...

Acknowledgements

First and foremost I would like to acknowledge my family, for their continued support and belief in me. Family is one of nature's masterpieces. Your convictions have upheld me in the days of doubt. I thank you from the bottom of my heart.

Secondly I would like to acknowledge my supervisor Hugh Murrell. I thank you for your patience and your advice and recommendations towards the delivery of this thesis.

I would also like to acknowledge Mr. Shaun Moodley from the Department of Computer Science UKZN for his support in resolving implementation issues with the java technology underlying the MPPSoft prototype package.

Software Note

The URLs listed in this thesis are for examination purposes only. To receive the full PyroSim, MPPAlign and MPPErrorCorrect Mathematica 8.0 packages as well as software manuals, please email me at anisaragalo@gmail.com.

Contents

List of Figures

LIST OF FIGURES

Glossary of MPP terminology

Base calling: The process of translating each MPP flowgram in a list of flowgrams into the most likely underlying nucleotide sequence.

Flow cycle: The flow of the distinct nucleotide bases over a template DNA sequence in the order T-A-C-G.

Flow-space: A domain in which the raw data inputted into functions (for MPP data analysis) is in the form of MPP flowgrams. Further, in flow-space a reference genome nucleotide sequence is embodied by a flow-space representation of itself.

Flow-space representation: A run length encoding of a DNA nucleotide sequence that is padded with homopolymers of length 0 in order to obey the order of nucleotides in contiguous T-A-C-G flow cycles.

Flow value: A real number that represents the light signal emitted when a specific nucleotide is flowed over a homopolymer during MPP. An observed flow value is directly proportional to the length of the underlying homopolymer.

Flow value specification: A flow value specification consists of:

- $\mathbf{b}$, the nucleotide base for which a flow value was observed
- $\mathbf{f}_b$, the observed flow value

A full flow value specification is therefore represented as $\{\mathbf{b},\mathbf{f}_b\}$. Note that every flow value specification has an underlying homopolymer specification.

Flowgram: A sequence of observed flow value specifications outputted as a result of MPP. The sequence of flow value specifications obeys the order of nucleotides in contiguous T-A-C-G flow cycles. A flowgram is a class of MPP read. Note that in this thesis, we sometimes refer to a Flowgram as a **Flow Value Progression.**

Flowgram graph: A graphical representation of a flowgram. A flowgram graph is comprised of color-coded vertical bars, whereby a specific color represents a specific nucleotide base. Further, each bar represents a flow value specification whereby the height of the bar is directly proportional to the observed flow value.

FVDM: (Flow Value Distribution Model) - A sequence of **N** distinct flow value distributions, where **N** is the maximum homopolymer length catered for by the model. The $\mathbf{N}^{th}$ distribution in the model shows the variation in the flow values that MPP yields for homopolymer length **N** according to the model

Homopolymer: A series of two or more contiguous identical DNA nucleotides (e.g. AAA or CCCC) that occurs within a DNA sequence.

Homopolymer progression: A sequence of homopolymer specifications that is padded with homopolymers of length 0 in order to obey the order of nucleotides in contiguous T-A-C-G flow cycles.

Homopolymer specification: A homopolymer is specified by:

- **b**, the nucleotide base comprising the homopolymer
- $\mathbf{n}_b$, the number of copies of **b** in the homopolymer.

A full homopolymer specification is therefore represented as $\{\mathbf{b},\mathbf{n}_b\}$

MPP: (Massively Parallel Pyrosequencing) - A DNA sequencing technique based on pyrosequencing technology. MPP involves the concurrent sequencing of multiple DNA fragments. Sequencing is conducted by contiguously flowing DNA nucleotides in the order T-A-C-G over the DNA fragments inside the MPP sequencing instrument and

observing the light energy emmited for each flow. The sequencing reads obtained by this sequencing methodology are characteristically short and error prone.

MPP read: A generic term that describes information about an underlying template DNA sequence. An MPP read is outputted as a result of an MPP experiment. The term encapsulates both sequencing reads and MPP flowgrams.

Quality score: A score assigned to each nucleotide in a sequencing read. The quality score of a nucleotide indicates the probability that the homopolymer length outputted by base calling at the nucleotide's position is correct.

Reference sequence: A nucleotide sequence extracted from the same genomic region (of the same organism) as the template DNA sequence whose genetic composition an MPP experiment seeks to determine. The reference sequence is used to interpret the reads derived from the MPP experiment.

Sequence-space: A domain in which the raw data inputted into functions (for MPP data processing) is in the form of MPP sequencing reads. In sequence-space, a reference genome sequence is embodied by a nucleotide sequence.

Sequencing read: A DNA nucleotide sequence outputted as a result of MPP. Ideally, this nucleotide sequence is an exact complement of a subsequence of the template DNA sequence. However since MPP is prone to sequencing error, a sequencing read contains errant nucleotide bases introduced by the sequencing technique. A sequencing read is a class of MPP read.

Template DNA: A DNA nucleotide sequence that is inputted into a MPP instrument. Given a template sequence, the aim of MPP is to determine the exact sequential ordering of nucleotide bases in the template DNA sequence.

454 Life Sciences: The biotechnology company behind the MPP sequencing methodology. MPP sequencing is also referred to 454 sequencing.

1

Introduction

This thesis aims to explore the use of computational algorithms in order to resolve the authentic variation of genome sequences found in Massively Parallel Pyrosequencing (MPP) data.

MPP is a sequencing methodology that generates large quantities of sequencing information/data. 454 life sciences implements this sequencing technology and as such it is often referred to as 454 sequencing.

MPP has a variety of applications. However we are specifically interested in experiments where MPP technology is used to determine the genome sequences present in a heterogeneous population.

In an experiment to sequence a heterogeneous population the interest typically lies in a particular genomic region. The sequencing data is therefore derived from a homologous genomic region and the diversity in the sequencing data reflects the diversity in the population.

MPP is however prone to sequencing errors. Sequencing errors are a secondary source of variation/diversity in sequencing data, the primary source being authentic variation in the population. Errors associated with this sequencing methodology occur in the form of the sequencing reads containing insertions, deletions or substitutions of

nucleotide bases with respect to the actual genome being sequenced [1].

Typically, in an experiment to sequence a diverse population, there is a reference genome derived from the same homologous genomic region of interest. According to Zagordi et al. [2], if sequencing data shows variation with respect to this reference genome there are two possibilities that should be considered:

1. A variant sequence with this variation is actually present in the population that we are sequencing.

2. The variation from the reference genome is due to technical noise (Errors conferred by the MPP methodology itself).

Computational and statistical challenges arise in the interpretation of MPP data to make this distinction between authentic variation and technical noise/sequencing error. Inferring the true variation in the underlying population from MPP data is in fact a complex undertaking[1].

The complexity of inferring authentic variation from MPP data is compounded by the fact that the error rate of this sequencing methodology (about 5-10 errors per 1000 base pairs) is characteristically higher than that of several existing sequencing methodologies. For instance direct PCR Sanger sequencing has a much lower error rate of about 0.01 errors per 1000 base pairs [3]. Note that here the term base pair refers to two nucleotides on opposite complementary DNA or RNA strands. The term kilobase (kb) is used to indicate 1000 base pairs. Therefore we can state the error rate of MPP to be 5-10 errors per kb.

In addition to the errors conferred on the reads by the sequencing methodology, pyrosequencing reads are characteristically short. According to Saeed et. al. [4], as of

[1]In the context of sequencing methodologies, the term "underlying population" refers to the population of DNA sequences that produced the sequencing data being analyzed. Inferring the underlying population from sequencing data is also referred to as inferring authentic variation from sequencing data. This inference is made by attempting to characterize the distinct genetic variants in the population that produced the sequencing data.

2008, pyrosequencing reads had an average length of 100-250 nucleotides and were much shorter than Sanger sequencing reads which had read lengths of 800-1000 nucleotides. Given a short sequencing read, it becomes a challenge to determine exactly where in the genome the read originated from. Successful reconstruction of the genome (or population of genomes) being sequenced requires that each sequencing read is placed in the correct context, in terms of which position in the genome it originates from.

The short lengths of pyrosequencing reads, in addition to sequencing errors, further compound the difficulty of inferring an underlying population from MPP data.

According to Eriksson et al. [3], given MPP data, a full characterization of the distinct variants in the underlying population consists of:

1. Deriving the set of possible underlying variants consistent with the data. This is a derivation of the possible distinct nucleotide sequences underlying the data.

2. Estimating the relative quantities/proportions in which the underlying variants exist.

We however limit the scope of this thesis. Given MPP data from a homologous genomic region and a reference genome (extracted from the same homologous genomic region), characterization of the underlying variants is restricted to determining the authentic variation **at each nucleotide position** relative to the reference genome. This is done in two stages:

1. **Stage 1**: The alignment stage - Each MPP read is placed in the correct context (in terms of its origin) with respect to the reference genome.

2. **Stage 2**: The error correction stage - Given a nucleotide position in the reference genome to which MPP reads have been aligned, authentic variation at the nucleotide position is distinguished from sequencing error.

This thesis aims to compare existing computational and statistical approaches to

MPP read alignment and error correction. Several algorithms implementing these approaches are described in literature [2, 3, 4, 13, 15, 17].

Our comparison of distinct algorithmic approaches to read alignment and error correction is based on two aspects:

1. A qualitative analysis of the alignment or error correction results.

2. A determination of the relative computational complexity and scalability of the algorithms with the same niche (be this read alignment or error correction).

In order to compare the algorithmic approaches in terms of complexity and scalability, we implement them on the same computing platform. This is in order to ensure that factors such as the time taken to compile an algorithm and the execution speed of the kernel running the algorithm are uniform and do not bias our results.

In order to test and compare the algorithmic approaches, we implement an MPP simulator based on the approach to MPP simulation by Balzer et. al. [7]. The simulator is able to read in a population of sequences and generate simulated MPP data. This facilitates our analysis of the algorithmic approaches to MPP read alignment and error correction, because we have the original population from which the simulated sequencing data is derived. The authentic variation at each nucleotide position is estimated by applying the algorithms to the simulated MPP data and should, in an ideal world, be identical to the variation (at the nucleotide position) in the population input into the simulator.

With this in mind, our study consists of two principal stages:

1. The design and implementation of a MPP simulator:

   The purpose of this is to simulate the production of MPP data from an underlying population of mixed proportions of variant sequences.

2. The analysis and comparison of algorithmic approaches to resolving the true authentic variation from the simulated MPP data. We conduct this by implementing selected computer algorithms (described in literature) on the same computing platform. We then carry out experiments in order to perform qualitative comparisons on the algorithms.

In addition to this, an important deliverable of this project is MPPSoft, a software prototype for the simulation as well as alignment and error correction of **small** quantities of MPP data. We implement this package as a java applet that is publicly accessible at the URL: `http://hammer.cs.ukzn.ac.za/~anisa/`.

The complete package consists of a java applet web interface that communicates with a Mathematica 8.0 back-end engine.

We opted for a java front-end because of the portability of java applications. Further, we opted for a Mathematica back-end because the Mathematica computing platform facilitates concise functional programming as well as function visualization.

The full implementation details of this software package are provided in chapters 4, 5 and 7 of this thesis.

We begin the conduction of this study with a thorough description of MPP technology, as well as the nature and cause of the errors that accompany this methodology.

The subsequent chapters of this thesis are organized as follows:

Chapter 2 is a detailed description of MPP technology as well as the source and nature of the errors that characterize it.

Chapter 3 consists of a literature review on the simulation of MPP as well as a review of the proposed algorithms to resolve the characteristics of the underlying population from MPP data. We review different MPP read alignment and MPP read error correction algorithms.

Chapter 4 discusses our approach to MPP simulation, based on a simulation approach proposed by Balzer et. al. [7]. We implement MPP simulation in a Mathematica

8.0 simulation package named PyroSim.

Chapter 5 describes our implementation of a selected array of MPP read alignment algorithms. It also describes how we verify that our implementations of the algorithms are correct. We mention the complexity of our implementation of each algorithm and draw a comparison between the computational complexities of the distinct algorithms.

Chapter 6 details the experiments that we conduct in order to compare the qualities of the distinct solutions produced by the distinct MPP read alignment algorithms. We also provide the results of our experiments and make conclusions based on the results.

Chapter 7 describes our implementation of a selected array of MPP error correction algorithms. We verify that our implementations of the algorithms are correct and compare their computational complexities.

Chapter 8 details the experiments that we conduct in order to compare the qualities of the distinct solutions produced by the distinct MPP error correction algorithms. We provide the results of our experiments and make conclusions based on the results.

Chapter 9 provides a conclusion on the findings of our qualitative experiments.

2

Background

This chapter seeks to provide a clear understanding of MPP technology. The sections of this chapter are organized as follows:

Section 2.1 discusses the principles underlying MPP.

Section 2.2 provides an example of the importance of the utilization of MPP in sequencing a heterogeneous genomic population.

2.1 The theory behind MPP

MPP is based on pyrosequencing technology. In order to have an understanding of the former it is vital to understand the fundamental technology. This section is organized as follows: Subsection 2.1.1 discusses the principle of pyrosequencing. Following this, Subsection 2.1.2 describes how MPP utilizes pyrosequencing technology. Subsection 2.1.3 discusses the accuracy and quality of MPP and subsection 2.1.4 determines the source of the technical noise (errors) that result from this sequencing methodology. Subsection 2.1.5 describes the interpretation and analysis of MPP data.

2.1.1 Principle of Pyrosequencing technology

All organisms store genetic information in the form of either deoxyribonucleic acid (DNA) or ribonucleic acid (RNA). Both of these nucleic acids are essentially long chains of building blocks referred to as nucleotides (nucleotide bases). The nucleotide building blocks in DNA are adenine (A), guanine (G), thymine (T) and cytosine (C). The same applies to RNA, except uracil (U) takes the place of thymine.

A nucleic acid strand is a sequence of nucleotides bound together by a backbone of strong covalent bonds. DNA is double stranded, whereas RNA is single stranded. In the case of DNA, the two strands that comprise its structure are complementary strands. The word complementary refers to a unique nucleotide base pairing mechanism. Each nucleotide base in a DNA strand can only pair with a specific nucleotide base in the complementary strand. Specifically, adenine pairs with thymine, whereas cytosine pairs with guanine.

DNA and RNA synthesis involve the action of a polymerase enzyme on a template (DNA or RNA) strand. The polymerase attaches complementary nucleotide bases to a growing complementary strand. Each incorporation involves the attachment of a nucleotide base (to the growing strand) that is a complement to the nucleotide base within the template strand at the specific position of the enzyme. The enzyme then moves along the template strand to the next nucleotide and repeats the process. This is called a nucleic acid polymerization reaction and the incorporation of a nucleotide to the growing complementary strand releases inorganic pyrophosphate (PPi) [5].

The pyrosequencing technique utilizes the principle of sequencing by nucleic acid synthesis and relies on the detection of pyrophosphate released from each nucleic acid polymerization reaction.

The technique involves utilizing a DNA template strand from the genome to be sequenced. In the case where the genetic information being sequenced is in the form of RNA (for example if we are sequencing a viral genome), complementary DNA (cDNA)

is generated from the RNA. Complementary DNA can be defined as DNA derived from RNA that serves as the template strand for the sequencing reaction.

An array of enzymes (DNA polymerase, ATP sulfurylase and luciferase) and reagents is mixed in with the template DNA and the four DNA nucleotides are flowed cyclically over the reagent mixture. Multiple flow cycles are repeated contiguously.

A single flow cycle is the flow of the distinct DNA nucleotides in the specific progression T-A-C-G. At any one of the four stages in a single flow cycle, there is only one type of nucleotide present in the mixture of reagents and enzymes. This is ensured by intermediately removing any unincorporated nucleotides after each distinct type of nucleotide is flowed within a cycle. Therefore at stage one of any flow cycle T is the only nucleotide present in the reagents, at stage two of any flow cycle A is the only nucleotide present in the reagents, at stage three of any flow cycle C is the only nucleotide present in the reagents and at stage four of any flow cycle G is the only nucleotide present in the reagents.

There are two types of pyrosequencing: solid-state pyrosequencing and liquid-state pyrosequencing. In solid-state pyrosequencing the intermediate removal of unincorporated nucleotides is achieved by a washing step in between flows, whereas liquid-state pyrosequencing includes a fourth enzyme in the reaction mixture, Apyrase, which breaks down the excess nucleotides in between nucleotide flows.

When the type of nucleotide flowed into the reagent and enzyme mixture is complementary to the template strand, nucleic acid polymerization occurs. This results in the release of inorganic pyrophosphate. The critical idea of pyrosequencing technology is the detection of nucleotide incorporation by detecting pyrophosphate release. This is achieved by converting the pyrophosphate released from the polymerization reaction to Adenoside Tri-Phosphate (ATP) through the action of ATP sulfurylase. The ATP then provides energy to the enzyme luciferase, which oxidizes luciferin to generate visible light. The set of reactions described is illustrated in Figure 2.1.

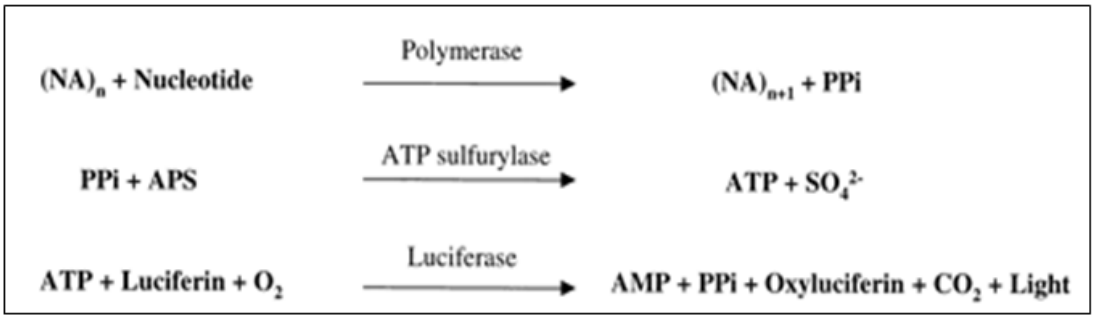

Figure 2.1: Pyrosequencing chemistry - The general principle behind pyrosequencing technology. Taken from [5]

Since the particular nucleotide in the reagent mixture is known at each stage of the cycle, light emission signals the incorporation of a known nucleotide.

The template strand may contain a sequence of contiguous identical nucleotide bases (i.e. a contiguous sequence of nucleotide bases that are all of one type, for example GGGGG or AA or TTTT). This contiguous sequence is referred to as a homopolymer.

In this scenario, the polymerase will similarly incorporate a sequence of contiguous identical nucleotide bases to the growing complementary strand. Therefore when the particular type of nucleotide that is complementary to the contiguous sequence in the template strand is flowed into the reagent mixture, multiple identical nucleotides are incorporated into the growing strand.

The amount of pyrophosphate released is always directly proportional to the number of incorporated nucleotides. This translates to the amount of visible light signal generated in turn being directly proportional to the number of nucleotides of a distinct type that are incorporated into the growing DNA chain. According to Marguiles et. al, this proportionality is maintained up to homopolymers of length eight [6].

Pyrosequencing measures the light signal emitted for each nucleotide flowed within each T-A-C-G flow cycle for a succession of flow cycles. The light intensity values generated for each nucleotide flow are also referred to as flow values.

The term flowgram is used to define a progression of flow values generated from

sequencing a template DNA sequence.

For the remainder of this thesis, we will use the terms flowgram and flow value progression interchangeably.

Flowgrams are well illustrated by flowgram graphs, which are outputted by pyrosequencing instruments. Figure 2.2 is an example of a flowgram graph.

A flowgram can subsequently be translated into a sequencing read [1].

Ideally, the sequencing read should be the exact complement of the underlying template strand. This is not the case, however, as sequencing errors (technical noise) may lead to augmented or diminished flow values, from which the underlying homopolymer lengths may be over-estimated or under-estimated respectively. The errors associated with pyrosequencing technology are discussed in more detail in subsections 2.1.3 and 2.1.4.

2.1.2 Massively Parallel Pyrosequencing - MPP

MPP is a large scale parallel implementation of pyrosequencing technology.

The sequencing methodology involves isolating and fragmenting the particular genomic region intended to be sequenced. Each genomic fragment is then separated into single strands, one of which is used as a template for sequencing. The template strand fragments are immobilized onto beads under conditions that constrain on average one unique fragment strand per bead. Subsequently the beads undergo PCR amplification in an emulsion, such that each bead will carry millions of copies of a unique DNA fragment strand [6].

Each bead is then deposited into its own well within a fibre-optic slide containing multiple wells. Small beads that contain a mixture of DNA polymerase, ATP sulfurylase and luciferase are also deposited into each well.

[1] A sequencing read is a sequence of nucleotide bases generated by translating a flowgram into a sequence of homopolymers. This is achieved by estimating the homopolymer length that produced each flow value. This is referred to as base calling and is discussed in detail in subsection 2.1.5.2.

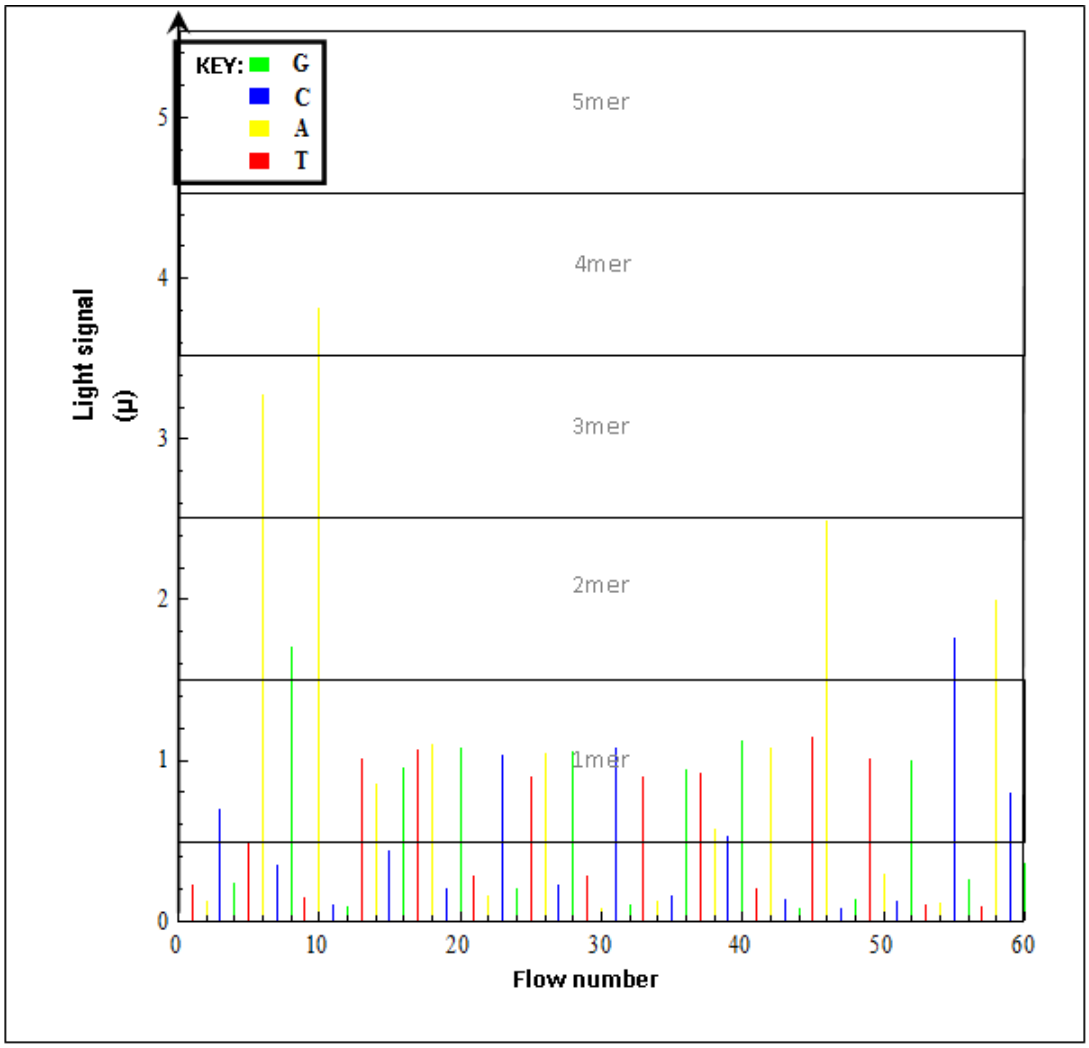

Figure 2.2: A Pyrosequencing Flowgram graph drawn in Wolfram Mathematica 8.0 - The flowgram graph indicates the light signal intensities produced for each cyclical flow of nucleotides. This translates to an indication of the number of nucleotides incorporated in each cyclical flow. In the above flowgram graph the ranges of light signals that correspond to particular homopolymer lengths are mapped out e.g. light signals in the interval 0.5 to 1.5 are shown to indicate the incorporation of a single nucleotide (which the diagram refers to as a 1mer). Similarly the region labeled as 2mer indicates a range of flow values that signal the incorporation of two nucleotides, and so on and so forth for the 3mer, 4mer and 5mer regions. Following the color key, the above flowgram graph could therefore be interpreted to identify the complementary sequence: CAAAGGAAAATAGTAGCTAG

The slide containing the reagents is then placed inside the sequencing instrument, which sequentially flows the four nucleotides over the wells in repetitive cycles. Each cycle is called a flow cycle and consists of the four distinct nucleotides being flowed in the order T-A-C-G.

MPP therefore involves the concurrent sequencing of the genomic fragments in wells, such that each well emits a light signal during nucleotide incorporation proportional to the number of incorporated nucleotides. Hence the sequencing instrument sequences multiple template strand fragments simultaneously.

454 sequencing technology has improved over the years. The original 454 sequencer, the GS 20 had the capacity to sequence an average of 200,000 fragments simultaneously. Its successors, the GS FLX and GS FLX titanium can sequence 400,000 and 1 million reads respectively.

Figure 2.3 is an illustration of MPP technology taken from [6].

2.1.3 Accuracy and quality of 454 Sequencing

Previously we stated that in an ideal world, a sequencing read should be the exact complement of the underlying template sequence. We now investigate why this is not always the case.

The 454 sequencing methodology introduces errors into the sequencing reads. According to Balzer et. al., one of the main challenges in MPP is resolving the correct homopolymer lengths from flow values [7].

We established earlier that an underlying homopolymer of length $\boldsymbol{n}$ produces a mean flow value of value of $\boldsymbol{n}$ during sequencing. Let us take note of the word "mean" in this statement. The individual light intensity values emitted for homopolymer lengths do not equal exact integers (due to experimental error). The flow values that pertain to a homopolymer length $\boldsymbol{n}$ are in fact distributed about $\boldsymbol{n}$. This can be observed in the variability of the peak lengths in the flowgram in Figure 2.2.

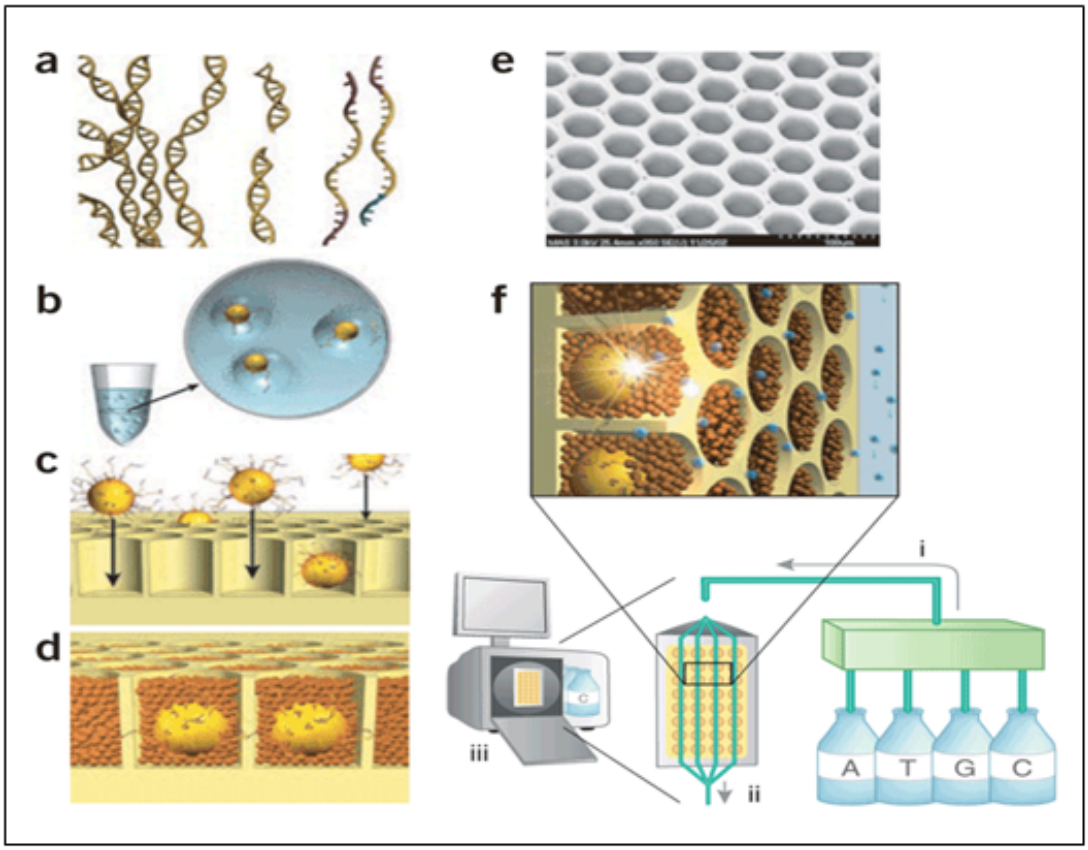

Figure 2.3: Massively Parallel Pyrosequencing - (a) The genomic DNA to be sequenced is isolated and randomly fragmented. Each fragment is separated into single strands. (b) Single stranded fragments are fastened to beads under conditions that constrain one fragment per bead. Each bead is sectioned into its own droplet of a PCR-reaction-mixture-in-oil emulsion. PCR amplification occurs inside each droplet such that all the beads carry ten million homogenous copies of a unique DNA template strand. (c) After de-emulsification the beads carrying multiple copies of single-stranded DNA templates are deposited into wells of a fiber-optic slide. (d) Miniature beads carrying the enzymes required for the cascade of pyrosequencing enzymatic reactions are deposited into each well. (e) Magnified fibre-optic slide, showing fiber-optic wells. (f) The 454 sequencing instrument is comprised of: a fluidic assembly (object i), a flow cell that includes the well-containing fiber-optic slide (object ii), a CCD camera that captures luminescence from the wells (part of object iii), and a computer supplying a user interface for the manipulation of the equipment (part of object iii). Taken from [6]

In the publication that provides the original description of MPP [6], Marguiles et. al. claim that the distribution of flow values that indicate a homopolymer of length $\boldsymbol{n}$ is Gaussian with a mean of $\boldsymbol{n}$ ($\mu = n$) and standard deviation directly proportional to $\boldsymbol{n}$ ($\phi \propto n$). Figure 2.4 is an example of the distribution of flow values that indicate a homopolymer of length 1 according to this model.

The primary cause of error in MPP is the fact that the distributions of flow values for the different homopolymer lengths overlap each other. This is shown in Figure 2.5. Balzer et. al. state that if there were no overlap in the distributions of flow values for different homopolymer lengths, MPP would produce error-free sequencing reads [7]. The standard deviations of the distributions increase with increasing homopolymer length resulting in greater overlap regions for higher homopolymer values (Figure 2.5).

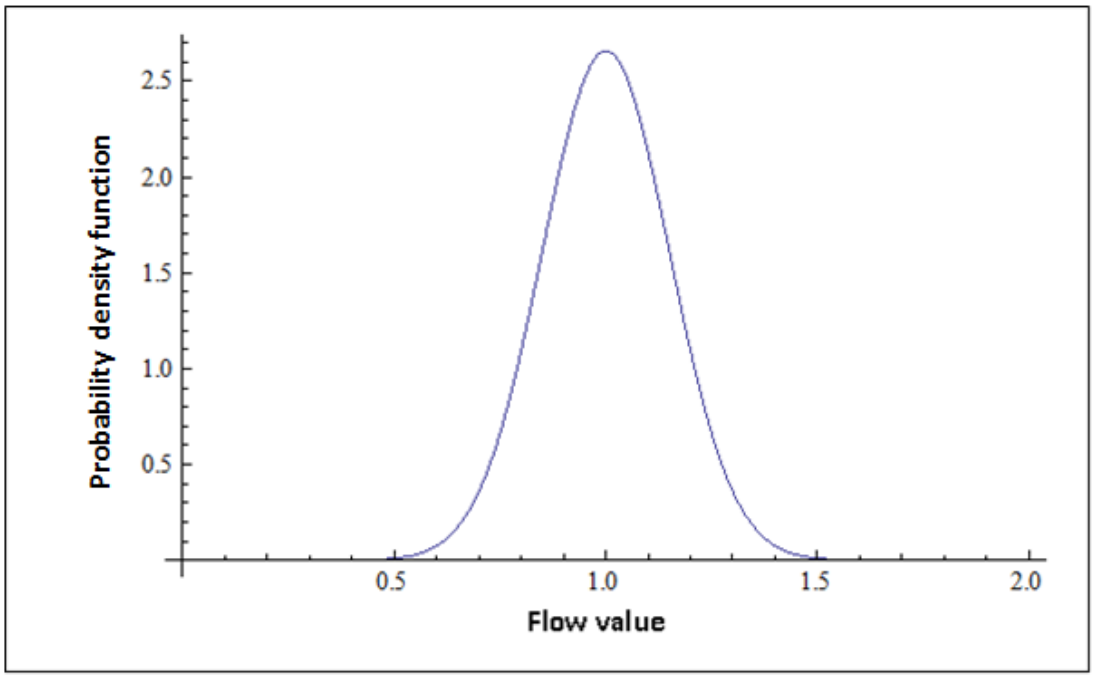

Figure 2.4: Distribution of flow values that indicate a homopolymer of length 1 drawn in Wolfram Mathematica 8.0 - Marguiles et. al estimate this distribution to be Gaussian with mean, $\mu = 1$, and standard deviation, $\sigma = k$, where k is a constant. We set $k = 0.15$ to produce the distribution shown in the figure.

Aside from the Gaussian model for the distribution of flow values about homopoly-

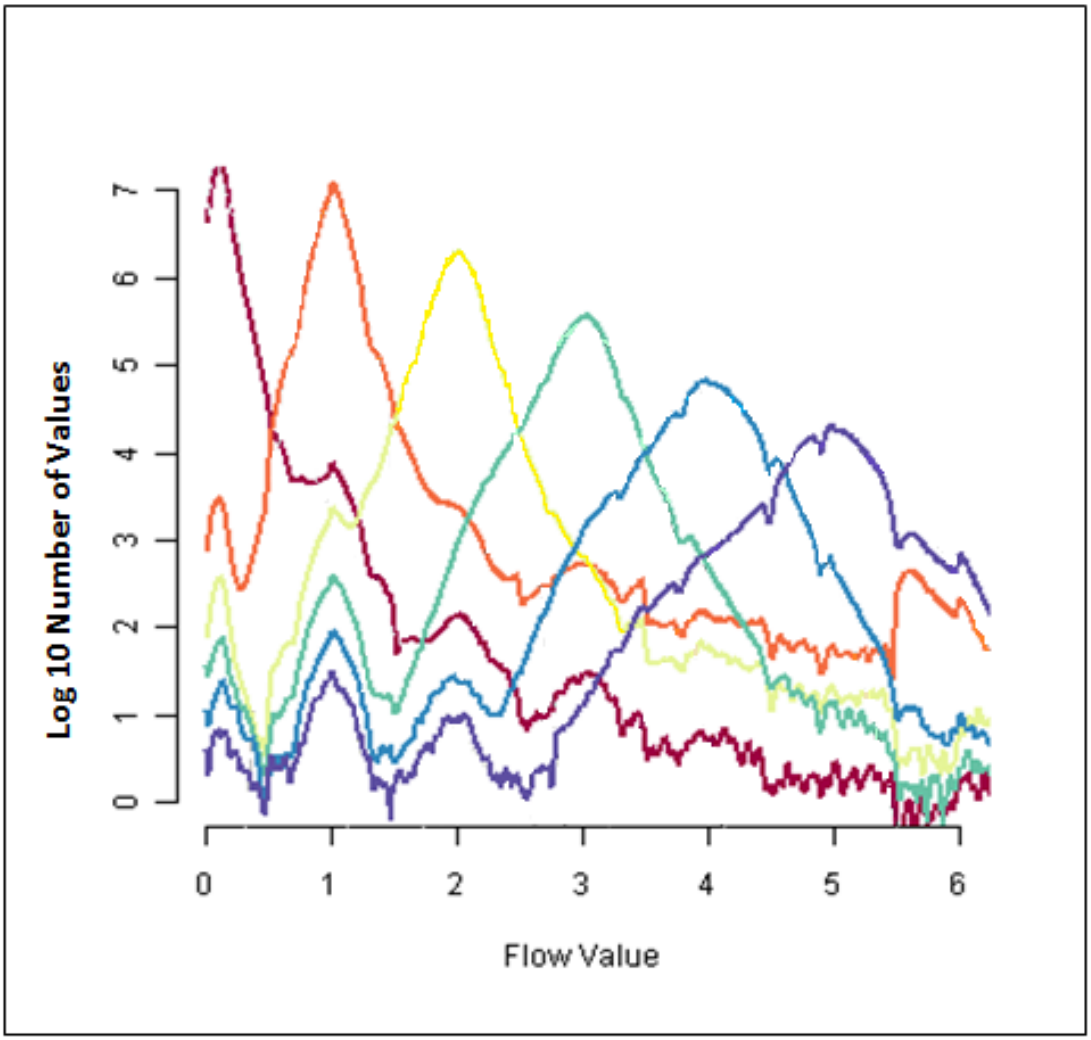

Figure 2.5: Overlapping flow value distributions for different homopolymer lengths - Taken from [7]

mer lengths provided by Marguiles et. al., there are other models for flow value distribution about homopolymer lengths. These models are discussed in detail in subsection 3.1.2. Among the different models proposed for the distribution of flow values about homopolymer lengths, there is consensus that the distributions of flow values for the different homopolymer lengths overlap.

Flow values that fall within overlapping regions are ambiguous, because it is not clear which homopolymer lengths they indicate. The ambiguity in flow values typically increases with the increase of homopolymer length. This is because of the increase in the overlaps of neighboring distributions with increasing homopolymer length [6, 7].

Ambiguity in flow values results in over-calls, where a flow value is assigned to a homopolymer length greater than the actual homopolymer that produced it, or under-calls, where a flow value is assigned to a shorter homopolymer length than in reality. Over-calls correspond to insertions, because the associated sequencing read will have an inserted base when compared to the actual genetic sequence, whereas under-calls correspond to deletions, as the read has one less base than the original sequence. When an over-call follows an under-call or vice-versa, this is perceived as a substitution with respect to the original sequence.

Negative flow values describe light signals whose intensity is too weak to indicate incorporation of bases. The distribution of negative flow values overlaps the distribution of flow values for a unit base incorporation. This means that if the flow value for a unit base incorporation is too weak, it can be perceived as a negative flow (under-call), whereas a high negative flow value can be perceived as a unit base incorporation (over-call). Negative flow values are also referred to as noise flow values. In the Margulies et. al. model, negative flow values follow a log-normal distribution.

Over-calls and under-calls due to homopolymer effects are the major source of error in pyrosequencing reads.

The errors resulting from MPP are not distributed evenly among the sequencing

reads. In an experiment conducted by Huse et. al [1], 82% of the reads had no errors, 93% had no more than a single error and 96% had no more than 2 errors. Also only a fraction of the reads, fewer than 2%, were responsible for about 50% of the errors in the entire set of reads.

2.1.4 The source of errors in MPP reads

In the publication that provided the original description of MPP [6], Marguiles et al conducted an experiment in which they sequenced Mycoplasma Genitalium DNA. The authors also created test fragments, which were intended to be difficult to sequence because they contained long stretches of homopolymers. Their intention in creating these test fragments was to assess the sequencing performance of the instrument. The individual test fragments were prepared and amplified by a different procedure to the standard preparation of template DNA for MPP.

They observed that the error rate that was produced for the test fragments was small compared to the error rate for the Mycoplasma Genitalium. This is in spite of the fact that the test fragment sequences were created such that they would result in several sequencing errors (Recall that the test fragments contained long homopolymer stretches and we have established the latter to be associated with higher sequencing errors).

They interpreted this observed inconsistency to indicate that the underlying sequencing technology in 454 sequencing, luminescence detection of nucleotide incorporation and the determination of flow values, is sound. This is because the test fragments were sequenced in the exact same way as the Mycoplasma Genitalium DNA and yet had a much lower error rate. The error in the latter must therefore have been introduced by the standard 454 sample preparation method (that was used to prepare the Mycoplasma Genitalium DNA).

The authors concluded that whereas the majority of errors in MPP do not result

from the core pyrosequencing technology, they are in fact a consequence of the experimental manipulation of sequences that takes place before the actual sequencing [6].

Let us recall that the preparation of the DNA to be sequenced involves its fragmentation and the subsequent binding of DNA molecules to beads in conditions that are intended to constrain one DNA fragment per bead. These beads are then each amplified, and each bead deposited in its own well of the PicoTitre plate prior to sequencing. Let us recall also that each well produces a sequencing read.

Marguiles et al suggest that the source of error in MPP technology is the binding of multiple DNA fragments to an individual bead prior to PCR amplification [6]. This results in a well carrying copies of more than one fragment (template) sequence. A well containing heterogeneous template fragments would frequently have different bases or varying homopolymer lengths at each position resulting in ambiguous flow values, neither fragment having enough luminescence to clearly register. The ambiguous flow values lead to errors when determining the underlying homopolymer length.

The phenomenon described above explains the disproportionate distribution in errors in the sequencing reads as it is likely that only a fraction of beads would have multiple templates binding to them. This would mean that only a fraction of the wells in the instrument would be heterogeneous.

Aside from heterogeneous wells, other sources of error in this sequencing methodology include insufficient flushing of nucleotides in between flows, resulting in carry forward events whereby single base insertions in the read occur relative to the genome being sequenced [6]. Also insufficient nucleotides in a flow can result in the unfinished extension of a homopolymer.

2.1.5 Interpretation and analysis of MPP data

MPP produces flowgrams, which indicate how many nucleotides are incorporated for each nucleotide flowed (within a flow cycle) for each well of the sequencing instrument.

Each flowgram can be plotted in a flowgram graph. A flowgram can also subsequently be translated into a sequencing read. A sequencing read is a DNA sequence that is the error containing complement of the template strand being sequenced.

A 454 sequencing instrument comes with accompanying software that produces flowgram graphs and performs a number of other processing steps: Quality Trimming, Base Calling and Quality Score Calculation. These steps are discussed in subsections 2.1.5.1, 2.1.5.2 and 2.1.5.3 respectively.

2.1.5.1 Quality Trimming

We have established MPP to be error prone. Marguiles et. al., however, provided a way of identifying high quality sequencing data [6]. This method involves identifying flow values that fall in the overlap region between the negative flow distribution and the unit base incorporation distribution. Light signals that range between 0.5 and 0.7 are identified to be within this region. Margulies et. al. found that if a considerable amount of the flow values in an MPP read fall in this interval, the MPP read is of poor quality. In this case poor quality translates to the read containing several errors with respect to the original genome. The authors claim that this correlation is because substantive values in the overlap region are an indicator of heterogeneous wells (beads).

This realization enabled Marguiles et. al. to develop a filter for selecting high quality read data. High quality reads were identified as reads in which the number of flow values in the overlap region were less than 5% of the total number of flows. In their experiment any reads that did not meet this standard were trimmed by eliminating flows starting from the end of the read until this standard is met. They however set a threshold for the minimum length of a read sequence such that if the sequence is

being trimmed is cut down to 84 flows (21 cycles) and still contains a proportion of intermediate values greater than 5%, the sequence is eliminated from the pool of high quality reads. The procedure described here is known as quality trimming.

Quality trimming can thus be defined as the trimming of a progression of flow values (emanating from a single well) such that the proportion of flow values that lie in the interval between 0.5 and 0.7 amounts to less than 5% of the flow values in the progression. Quality trimming filters out most reads that result from heterogeneous wells.

2.1.5.2 Base calling

The heterogeneous wells in which one template is very dominant are difficult to filter out of the pool of acceptable reads. This leads to the aforementioned overlapping distributions of light signals for different homopolymer lengths.

The resolution of homopolymer lengths from light signal intensities is referred to as base-calling. The number of bases called is not intuitive because of the ambiguity in light signals (due to the overlapping distributions for different homopolymer lengths).

The number of bases called is determined from the light signal (flow value) using Bayes theorem. This involves calculating the probability of each homopolymer length given the observed light signal according to Equation 2.1:

Equation 2.1:

$$P(n|s) = \frac{P(s|n) \times P(n)}{\sum_j P(s|j) \times P(j)}$$

- s - Observed light intensity signal (flow value)
- n - Length of the homopolymer that produced the signal

The probability $P(s|n)$ of measuring a signal $\boldsymbol{s}$, given that the underlying homopolymer length is $\boldsymbol{n}$ depends on the distribution model that is used to characterize the distributions of flow values for the different homopolymer lengths.

Recall that in subsection 2.1.3, we described a model for the distributions of flow values for the different homopolymer lengths proposed by Marguiles et. al. in [6]. According to this model the distribution of flow values/light intensity values that indicate a homopolymer of length $\boldsymbol{n}$ is Gaussian with a mean equal to $\boldsymbol{n}$ ($\mu = n$) and standard deviation directly proportional to $\boldsymbol{n}$ ($\phi \propto n$). Also the distribution of light intensity signals for negative flow values have a log-normal distribution.

Using this model, $P(s|n)$ is drawn from the gaussian/normal distribution or log-normal distribution that characterizes the distribution of flow values about $\boldsymbol{n}$. For example if $\boldsymbol{s}$=1.2 and $\boldsymbol{n}$=1, $P(1.2|1)$ is derived from the probability density function of the gaussian distribution for homopolymer length 1 (i.e. a gaussian distribution with $\mu = 1$ and $\phi \propto 1$).

Marguiles et. al state that for a sequence of nucleotides, the probability of encountering a homopolymer of length $\boldsymbol{n}$ (i.e. P ($\boldsymbol{n}$)) is $\frac{1}{4^n}$. Therefore taking P($\boldsymbol{n}$) to be the prior probability of homopolymer length $\boldsymbol{n}$, and computing the likelihood $P(s|n)$, we can obtain the posterior probability of a homopolymer length given the flow value $\boldsymbol{s}$.

Equation 2.1 is applied for all homopolymer lengths. Subsequently the homopolymer length that registers the highest probability given the observed light signal is taken to be the actual homopolymer length in the underlying sequence.

2.1.5.3 Quality Score Calculation

Each base in a pyrosequencing read is assigned a quality score value. According to Marguiles et. al., the quality score is a measure of the probability that the homopolymer length at a particular nucleotide position is correct [6].

For example, if a signal, $\boldsymbol{s}$, indicates that three A's should be called, the quality score

for the second A in the homopolymer is obtained from the probability of a homopolymer of length 2 or greater given the signal $\boldsymbol{s}$.

The quality score of a base in a read is obtained by Equation 2.2.:

Equation 2.2:

$$Q = -10 \times \log_{10}[1 - Pr]$$

- $\boldsymbol{Q}$ - Quality score
- $\boldsymbol{Pr}$ - Probability that the base at that position in the homopolymer is correct (Equation 2.3)

The probability of a base being correct, $\boldsymbol{Pr}$, is obtained from Equation 2.3.

Equation 2.3:

$$Pr = \sum_{j=n}^{\infty} P(j|s)$$

- $\boldsymbol{Pr}$ - Probability of the base at position $\boldsymbol{n}$ in the homopolymer being correct. This is equivalent to the probability of a homopolymer of at least length $\boldsymbol{n}$ given a flow value $\boldsymbol{s}$. $\boldsymbol{Pr}$ is obtained by summing of the posterior probabilities of homopolymers of with length $\boldsymbol{n}$ and greater, given $\boldsymbol{s}$.

Revisiting our example let us take the flow value s mentioned earlier that indicates the homopolymer AAA. From the above formulations, the quality score of the nucleotide A at position 2 is a measure of the probability of a homopolymer of at least length 2, given the flow value, $\boldsymbol{s}$. This is found by calculating the sum of the probabilities of

homopolymers of length 2 and above ($\boldsymbol{Pr}$). Our $\boldsymbol{Pr}$ metric then undergoes a conversion into the quality score $\mathbf{Q}$.

2.2 The use of MPP in sequencing heterogeneous HIV viral populations

This section provides an example of the use of MPP in sequencing heterogeneous populations. Specifically, example is a look at the use of this sequencing methodology in the estimation of the heterogeneous HIV viral population within an HIV infected patient.

The HIV viral population within a single infected individual is highly dynamic and is characterized by rapid virus production and turnover rates. According to Perelson et. al, the within-patient average HIV-1 production rate is estimated at 10.3×10^9 new within-patient virus particles per day [9].

Aside from rapid reproduction, another critical factor that contributes to the HIV population dynamics is a high mutation rate. The mechanism by which the HIV virus replicates, reverse transcription, is notoriously inclined to error. On average one mutation is introduced for each new viral genome transcribed [10]. This means that whenever a viral particle reproduces by copying its genetic material, each new virus produced has a genome that differs from the parental virus by a single nucleotide.

As a consequence of both of the above-mentioned factors, the within-patient viral population consists of a complex and highly heterogeneous mixture of different mutants (strains) of the virus. These strains exist in different relative proportions within the population. This phenomenon is described as a within-host viral quasi-species[1].

Anti-retroviral (ARV) drug treatment was introduced with the aim of curbing viral replication in a patient in order to contain the progression of the within-patient HIV virus and lengthen patient survival.

[1] A viral quasi-species is a diverse population of evolutionarily related viral strains [3] existing competitively within a single environment

The complexity of the viral quasi-species can however lead to a failure of ARV treatment. Early research established that the viral quasi-species is a clear obstacle in the control of viral diseases by drug therapy [11]. This is due to ARV drug resistance. A viral particle is resistant to a particular ARV drug if the presence of the drug does not inhibit (or curb) the particle's replication.

The resistance of a particular HIV strain to one or more ARV drugs is linked to specific genetic mutations in the HIV viral genome of the strain. A number of distinct mutations of the HIV virus that confer the resistance of strains to distinct ARV drugs have been identified [12]. Specifically, mutations within the HIV pol gene are associated with drug resistance [11].

In order to avoid ARV drug failure, it has become imperative to perform resistance testing on HIV patients prior to the prescription of ARVs[1].

A faster, cheaper and less laborious means of performing resistance testing is Genotypic testing.

Genotypic testing is a method of resistance testing that is conducted by using a sequencing method to determine the heterogeneous genetic sequences of the HIV viral population that exist in a sample of a patient's blood. From the distinct HIV variants identified by the sequencing method, drug resistant mutants (variants) can be distinguished. This is possible recalling the argument that specific mutations are directly linked to specific mechanisms of drug resistance.

MPP has been shown to be able to detect minority variant HIV drug resistant mutations in patients which are not detectable by other sequencing methods.[14]

By the example provided here, we have established the importance of the use of MPP in sequencing a homologous region (the pol gene) of a heterogeneous population (the HIV viral quasispecies). It is clear that, because of the sensitivity of experiments

[1]Resistance testing is the conduction of experiments with the aim of identifying the drug resistant HIV strains in a patient prior to the administering of ARVs. Following resistance testing, the ARVs that a patient is predisposed to resist are avoided in the treatment regimen, reducing the risk of drug failure.

such as this, accurate and precise determination of the characteristics of the underlying population from the sequencing data is required. We therefore see the importance of conducting a thorough analysis, in a study such as this, in order to determine the best algorithmic approach to determining the characteristics of the underlying population from MPP data.

3

Literature Review

In the introductory chapter of this thesis, we described our research as being composed of two primary objectives:

1. The simulation of MPP

2. The comparison (in terms of both the quality of output and computational complexity) of existing algorithmic approaches to MPP read alignment and error correction.

3.1 Simulation of MPP

The simulation of MPP aims to produce the same output as the instrument does in reality. Specifically, according to Balzer et. al., the aim of simulation is to confer errors onto sequencing reads. The errors conferred should be characteristic of this sequencing methodology [7].

We found only one discussion of the stages involved in MPP simulation in literature. The discussion is by Balzer et. al. The authors apply their approach to MPP simulation in their simulator, Flowsim [7].

According to Balzer et. al., the simulation of MPP is achieved by beginning with sequences that would be produced (in sequencing a DNA template) if the sequencing methodology were error-free. These ideal sequences are exact complement sequences of subsequences of the template sequence (we say subsequences here because we have established that MPP technology produces short reads of the template DNA sequence).

Balzer et. al. state that the errors characteristic of MPP are conferred onto these exact complement subsequences through the use of a flow value distribution model[7]. From now on we will use the letters FVDM to abbreviate the term flow value distribution model.

A FVDM is an estimation of the underlying distribution of flow values for each distinct homopolymer length.

The following subsections are organized as follows. Subsection 3.1.1 discusses the Balzer et. al. approach to MPP simulation and describes the use of FVDMs. Subsection 3.1.2 discusses FVDMs in detail and describes how they are generated. Subsection 3.1.3 discusses existing MPP simulators.

3.1.1 MPP Simulation Stages

We view the generic approach to the simulation of MPP proposed by Balzer et. al. as being composed of two fundamental stages.

3.1.1.1 Stage 1: Generating a progression of flow values from the complement of a subsequence of an underlying template DNA sequence

This stage in itself consists of two distinct sub-stages.

Let us take the sequence below to be the complement of a subsequence/fragment of an underlying template DNA sequence. Recall that this means that the sequence below represents an ideal sequence that would be observed in sequencing the template DNA if the sequencing methodology were error free.

We wish to view the changes to this subsequence through the two distinct sub-stages.

AAAGTAATTCCC ← Ideal sequence - Exact complement of a subsequence of the template DNA

1. **Sub-Stage 1: Deriving a homopolymer progression from a DNA fragment**: Recall that pyrosequencing involves the contiguous cyclical flow of nucleotides in the order T-A-C-G. We can therefore obtain a run-length encoding of our ideal sequence that is constrained by the ordering of a flow cycle:

   { {T,0},{A,3},{C,0},{G,1},{T,1},{A,2},{C,0},{G,0},
   {T,2},{A,0},{C,3},{G,0} } ← Run-length encoding of underlying DNA subsequence

   This run length encoding is referred to as a homopolymer progression. Note that a homopolymer progression is a list of contiguous homopolymer specifications where each homopolymer specification takes the form {DNA nucleotide, homopolymer length}.

2. **Sub-Stage 2: Deriving a flowgram from a homopolymer progression** Recall that a MPP instrument generates a light signal/flow value from each homopolymer length during the contiguous cyclical flow of nucleotides.

   We have established that the flow values that indicate a homopolymer length $\boldsymbol{n}$ are distributed about $\boldsymbol{n}$. Let us also recall that the distributions of flow values pertaining to the distinct homopolymer lengths overlap, leading to ambiguity in the flow values.

   In generating flow values, a MPP simulator should be able to model this ambiguity and the subsequent errors that result from it.

In order to achieve this, MPP sequencing simulation relies on a FVDM, such as the Margulies et. al. model discussed in the previous chapter. The varying degrees of the overlaps among the distinct flow value distributions for the distinct homopolymer lengths are determined by the FVDM. The degrees of the overlaps determine the magnitude of ambiguity in the flow values. The FVDM therefore directly determines the ambiguity in the flow values and by extension, the spread of errors in simulated sequencing reads.

According to Balzer et. al. [7], a homopolymer length can be converted to a flow value by randomly drawing a value from the distribution of flow values for that specific homopolymer length (according to the FVDM). A homopolymer progression can therefore be translated into a progression of flow values.

Let us once again take our example into account. If we use the Margulies et. al. FVDM, we recall that flow values generated from a homopolymer length $\boldsymbol{n}$ have a Gaussian distribution with $\mu = n$ and $\phi \propto n$ i.e. $\phi = k \times n$, where k is a constant. For the example let us set $k = 0.15$ so that the distribution of flow values for a homopolymer length n is Gaussian with $\mu = n$ and $\phi = 0.15 \times n$. Recalling that the Margulies et. al. model describes negative flow values as being log-normally distributed, let us set negative flow values to be log-normally distributed with $\mu = 0.23$ and $\phi = 0.15$.

Then going along our homopolymer progression:

- The homopolymer length 0 in **{T,0}** is converted into a flow value by drawing a random value from the distribution of flow values for homopolymer length 0 (i.e. log-normal distribution with $\mu = 0.23$ and $\phi = 0.15$.)
- The homopolymer length 3 in **{A,3}** is converted into a flow value by drawing a random value from the distribution of flow values for homopolymer length 3 (i.e. Gaussian distribution with $\mu = 3$ and $\phi = 0.15 \times 3$.)

- The homopolymer length 0 in **{C,0}** is converted into a flow value by drawing a random value from the distribution of flow values for homopolymer length 0 (i.e. log-normal distribution with $\mu = 0.23$ and $\phi = 0.15$.)
- The homopolymer length 1 in **{G,1}** is converted into a flow value by drawing a random value from the distribution of flow values for homopolymer length 1 (i.e. Gaussian distribution with $\mu = 1$ and $\phi = 0.15 \times 1$.)
- The homopolymer length 1 in **{T,1}** is converted into a flow value by drawing a random value from the distribution of flow values for homopolymer length 1 (i.e. Gaussian distribution with $\mu = 1$ and $\phi = 0.15 \times 1$.)
- The homopolymer length 2 in **{A,2}** is converted into a flow value by drawing a random value from the distribution of flow values for homopolymer length 2 (i.e. Gaussian distribution with $\mu = 2$ and $\phi = 0.15 \times 2$.) ...

and so on and so forth.

An example flowgram that could be generated from our progression of homopolymer lengths is:

{ {T,0.241},{A,2.889},{C,0.211},
{G,1.011},{T,0.986},{A,1.478},{C,0.001},{G,0.105},
{T,2.010},{A,0.123},{C,3.510},{G,0.198} } ← Example flowgram

3.1.1.2 Stage 2: Base Calling

The second stage of simulation is base calling, which is the translation of a flowgram (generated in stage 1) back into a homopolymer progression and subsequently a nucleotide sequence. Base calling was described in detail in the previous chapter of this thesis. Note that the base calling procedure also uses an FVDM

in its computations (specifically to find the value of $P(s|n)$ - refer to the previous chapter).

Because of ambiguous flow values, the homopolymer progression generated from performing base calling on a flowgram will not be identical to the original homopolymer progression that produced the flowgram.

Taking our example once again, the flowgram that we generated is once again shown below:

{ {T,0.241},{A,2.889},{C,0.211},
{G,1.011},{T,0.986},{A,1.478},{C,0.001},{G,0.105},
{T,2.010},{A,0.123},{C,3.510},{G,0.198} } ← flowgram

By base calling, this flowgram can be interpreted to have been generated from the homopolymer progression:

{ {T,0},{A,3},{C,0},{G,1},{T,1},{A,1},{C,0},{G,0},
{T,2},{A,0},{C,4},{G,0} } ← homopolymer progression

which translates to the DNA subsequence:

AAAGTATTCCCC ← sequencing read

This sequencing read contains a deletion of the base A and an insertion of the base C with respect to the original ideal sequence AAAGTAATTCCC.

The Balzer et. al. approach to simulation is based on the fact that because the errors in the sequencing read above were generated through the use of a FVDM that characterizes MPP, they are typical/characteristic of MPP.

In a follow up paper Balzer et. al. expand their approach to MPP simulation. The authors enable their simulator to read in a single genome reference

sequence and subsequently generate reads from several copies of variants of the input sequence[8]. Further, the authors verify that the insertion and deletion errors in sequencing reads that occur due to homopolymer effects contribute significantly to the errors inferred by MPP.

3.1.2 Flow value distribution models (FVDMs)

In literature, FVDMs for MPP are generated by sequencing two or more known genomes [6, 7, 15, 16, 17]. Specifically, when a known genome is sequenced, each flowgram (progression of flow values) obtained is matched to its parent sub-sequence (i.e. the subsequence it originated from) within the genome.

From this matching, the homopolymer lengths that are responsible for the different light intensity signals (flow values) emitted are clear. According to Quince et. al. normally up to 10,000 flow values are observed for a given homopolymer length [15].

For each homopolymer length we then plot a histogram of the flow value distribution pertaining to the specific homopolymer length. According to Balzer et. al., given a specific homopolymer length, the distributions of flow values for the four distinct DNA nucleotide types (T, A, C and G) for that homopolymer length are very similar [7]. The authors therefore combine the observations of flow values for the four distinct nucleotides pertaining to a specific homopolymer length into a single distribution for that homopolymer length.

Given histogram data (for each homopolymer length), a FVDM is generated by estimating the probability density function of the flow value distribution for each homopolymer length [6, 7, 16, 17].

We have defined an FVDM as a parametric/non-parametric estimation of the underlying distribution of flow values for each homopolymer length. For higher homopolymer lengths for which the number of flow values (pertaining to the specific homopolymer lengths) observed is small, a parametric distribution may be obtained by extrapolating

from the distributions of lower homopolymer lengths [17].

We have identified four distinct FVDMs in literature:

1. The Margulies et. al. FVDM
2. The Richter et. al. FVDM
3. The Quinlan et. al. FVDM
4. The Balzer et. al. FVDM

Given a homopolymer length $\boldsymbol{n}$, the listed FVDMs propose different shapes for the distribution of flow values pertaining to $\boldsymbol{n}$.

The difference in the proposed FVDMs is due to a difference in the observations in the MPP experiments conducted by authors. This difference in observations may be due to a number of factors.

Previously, we established that the sample preparation step prior to MPP sequencing directly influences the ambiguity in MPP flow values. By extension, sample preparation directly influences the shape of the MPP FVDM homopolymer length distributions. Insufficient flushing of nucleotides between flows and carry forward events are also influences on the shape of the FVDM distributions.

We surmise that MPP experiments are very sensitive and therefore systematic errors may be the primary cause of the differences in observed FVDMs. In the conduction of MPP experiments, the experimenters, equipment and experiment environments differ. Also there is a difference in the known genomes that are sequenced.

Subsections 3.1.2.1, 3.1.2.2, 3.1.2.3 and 3.1.2.4 describe the Margulies, Richter, Quinlan and Balzer FVDMs respectively.

3.1.2.1 Margulies et. al. FVDM

In the publication that provided the original description of MPP, Margulies et. al. estimated that negative flows (flow values pertaining to homopolymer length 0) follow

a log-normal distribution, while all positive flows are normally distributed with mean approximately equal to the underlying homopolymer length ($\mu \approx n$)and standard deviation directly proportional to the underlying homopolymer length($\phi \propto n$ or $\phi = k \times n$ where k is a constant)[6].

3.1.2.2 Richter et. al. FVDM

Similarly to Margulies et. al., Richter et. al. also propose a Gaussian flow value distribution model [16]. The latter model the distribution of a homopolymer of length $\boldsymbol{n}$ (for $n \geq 1$) as Gaussian with a mean $\mu = n$ and standard deviation $\phi = k \times \sqrt{n}$ where $\boldsymbol{k}$ is a fixed proportionality factor. They establish the value of $\boldsymbol{k}$ to be 0.15. Thus $\phi = 0.15 \times \sqrt{n}$ for the Gaussian distribution pertaining to homopolymer length $\boldsymbol{n}$. According to these authors, basic statistics imply that the standard deviation of the Gaussian distribution should be directly proportional to the square root of $\boldsymbol{n}$, hence the described formula for the standard deviation.

However let us once again recall that the original description of MPP by Margulies et. al. states that the standard deviation of the Gaussian distribution of a homopolymer of length $\boldsymbol{n}$ should be directly proportional to $\boldsymbol{n}$ and not the root of $\boldsymbol{n}$. In line with this Richter et. al. provide an alternative formulation for the standard deviation of the Gaussian distribution pertaining to homopolymer length $\boldsymbol{n}$: $\phi = 0.15 \times n$.

In the case of both alternative models for the Gaussian distributions for homopolymer lengths $\boldsymbol{n}$ where $n \geq 1$, the Richter et. al. model describes the distribution of negative flow values as log-normal with a mean $\mu = 0.23$ and a standard deviation $\phi = 0.15$.

Figure 3.1 is a depiction of both the described Richter et. al. FVDMs drawn in Wolfram Mathematica 8.0. In both models, note the overlaps in distributions for different homopolymer lengths. These overlaps are the source of flow value ambiguity according to the Richter et. al. FVDM.

From now on we will refer to the first model in Figure 3.1 as version 1 of the Richter et. al. model and the second model in Figure 3.1 as version 2 of the Richter et. al. model.

3.1.2.3 Quinlan et. al. FVDM

Quinlan et. al. describe a different model for the distribution of flow values about the distinct homopolymer lengths. The authors claim that for a given homopolymer length the distribution of flow values pertaining to the homopolymer length is best approximated by a non-central Students t distribution [17]. They however do not explicitly specify the parameters of the non-central Students t distributions for the distinct homopolymer lengths. Figure 3.2 is a depiction of how well non-central Student t distributions approximate flow value distributions (for distinct homopolymer lengths).

3.1.2.4 Balzer et. al. FVDM

Balzer et. al. evaluate both the normal/log-normal [6, 16] and the non-central Students t distribution FVDMs [17] and find that both of these models do not fit the distributions of flow values that they observed about the distinct homopolymer lengths [7]. The authors state that an alternative to these parametric models is the use of non-parametric empirical distributions for the distinct homopolymer lengths. They estimate the empirical distributions by sequencing known genomes.

Balzer et. al further state that for homopolymer lengths greater than 5, the flow values observed are sparse. The authors therefore estimate distributions of flow values for higher homopolymer lengths ($n > 5$) by extrapolating from estimated parametric distributions for the shorter homopolymer lengths.

The model suggested by the authors therefore includes non-parametric empirical distributions for homopolymer lengths 0 to 5 (for which sufficient data was observed) and parametric distributions for homopolymer lengths greater than 5 (for which insuffi-

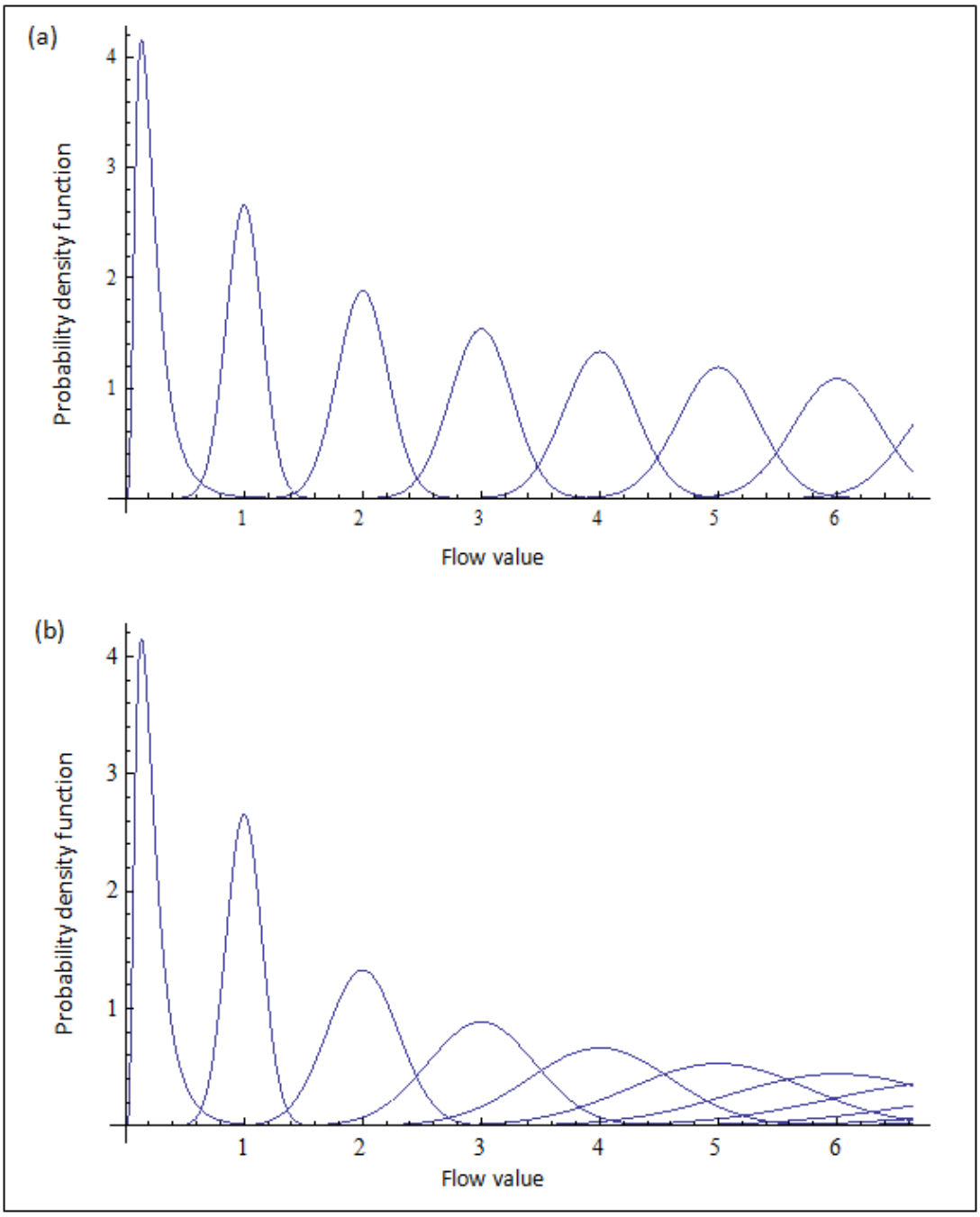

Figure 3.1: Richter et. al. flow value distribution models (showing distributions for homopolymer lengths 0 to 6) drawn in Mathematica 8.0 - (a) Given a homopolymer length $\boldsymbol{n}$ (where $n \geq 1$) the distribution of flow values for $\boldsymbol{n}$ is Gaussian with $\mu = n$ and $\phi = 0.15 \times \sqrt{n}$. (b) Given a homopolymer length $\boldsymbol{n}$ (where $n \geq 1$) the distribution of flow values for $\boldsymbol{n}$ is Gaussian with $\mu = n$ and $\phi = 0.15 \times n$.

For both flow value distribution models when $n = 0$ the distribution of flow values is log normal with $\mu = 0.23$ and $\phi = 0.15$.

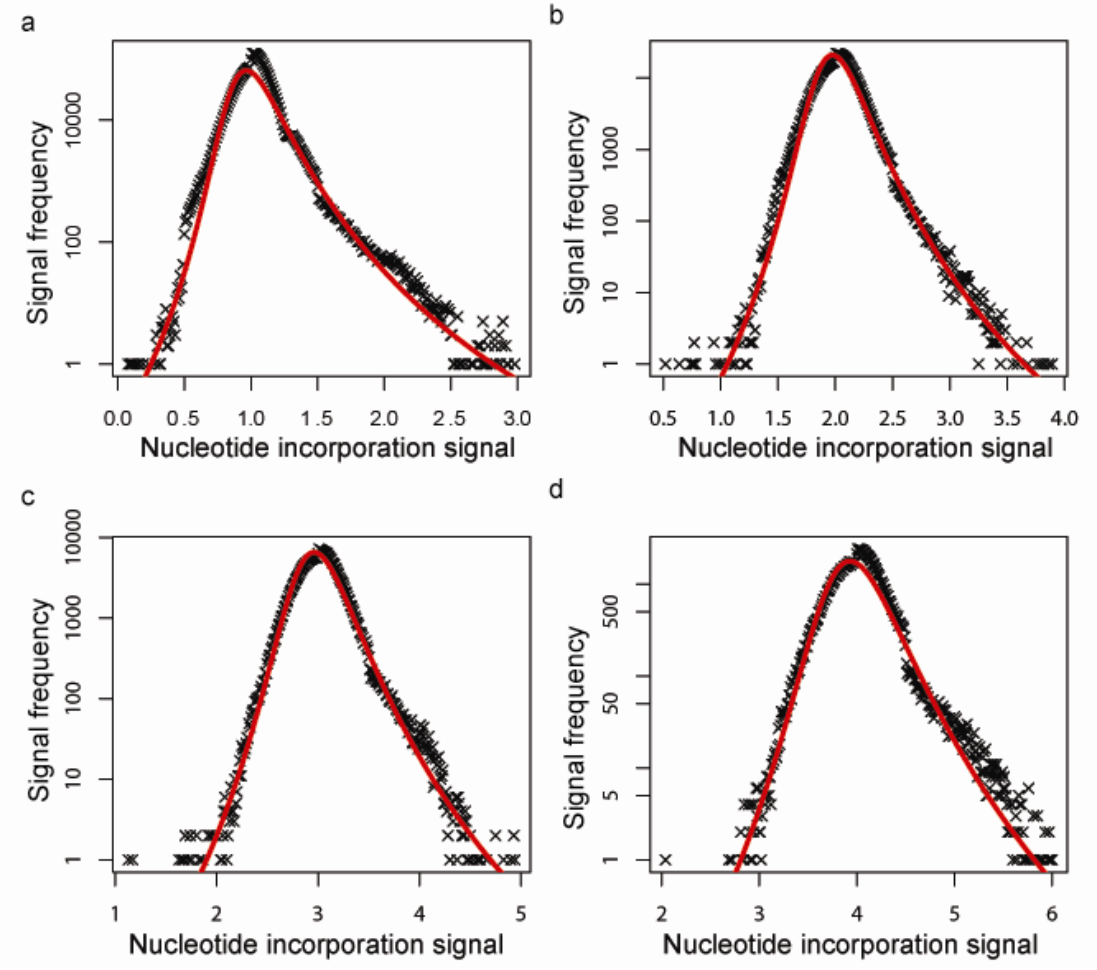

Figure 3.2: Flow value distributions for distinct homopolymer lengths observed by Quinlan et. al. - (a) Length = 1. (b) Length = 2 (c) Length = 3. (d) Length = 4. The observed flow values are shown in black. The approximating non-central Student's t distributions of flow values for the distinct homopolymer lengths are shown in red. Taken from [17]

cient data was observed and the distributions were therefore obtained by extrapolation from estimations of parametric distributions for homopolymer lengths 0 to 5).

Figure 3.3 is a diagram of the Balzer et. al. flow value distribution model.

3.1.3 MPP simulators

To our knowledge, there are not many existing simulators for 454 sequencing data.

In [16], Richter et. al describe a simulator named MetaSIM, designed for use in Metagenomics. MetaSIM is a generic sequencing simulator in the sense that it models different sequencing technologies.

MPP is among the sequencing technologies modeled by the MetaSIM sequencing simulator. The simulator employs the Richter et. al. FVDMs discussed in the previous subsection. MetaSIM produces an output of simulated pyrosequencing reads as Fasta files. This simulator does not output simulated flowgrams (the progressions of flow values that are the intermediate step in deriving simulated sequencing reads).

In [7], Balzer et. al. describe a simulator named Flowsim. Flowsim utilizes the Balzer et. al. FVDM.

According to the authors of Flowsim, there is a clear degradation in the accuracy of any read produced by this sequencing technology as one travels along the length of the read. In other words, the authors claim to have observed a degradation in the accuracy of flow values when comparing earlier flow cycles to later flow cycles in the sequencing reads obtained in their experiments.

They maintain that the flow position of a flow value has a noteworthy influence on its accuracy. This was observed in an experiment that compared read sequences to the original template sequences from which they were generated. In the experiment they measured the difference between a flow value and the homopolymer length underlying the flow value for each flow cycle. They observed a clear increase in the standard deviations of flow values from their underlying homopolymer lengths with the progression

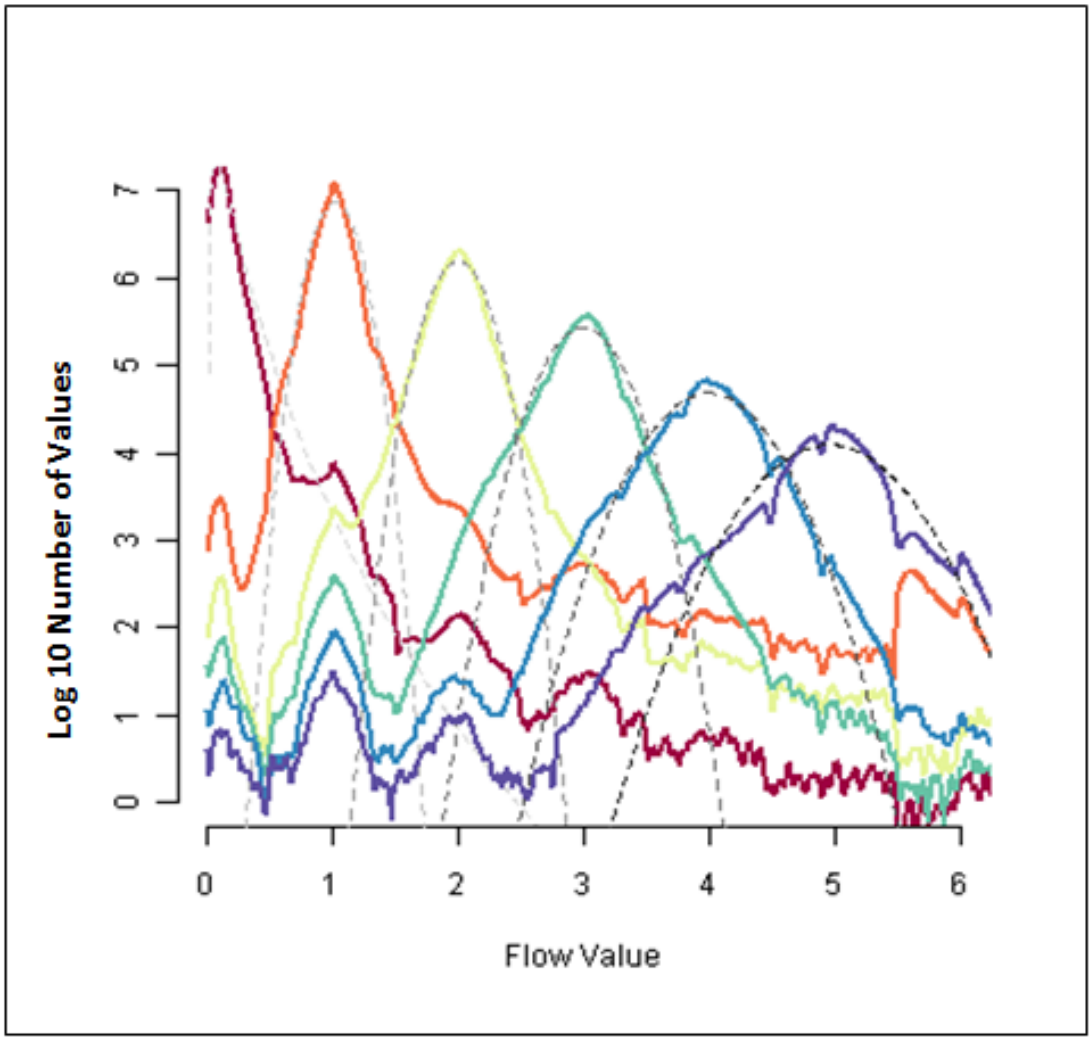

Figure 3.3: Balzer et. al. flow value distribution model. - The non-parametric empirical distributions of flow values for homopolymer lengths 1,2,3,4 and 5 are shown. Fitted parametric (log-) normal distributions are shown in gray. The fitted parametric distributions are used to extrapolate distributions for homopolymer lengths greater than 5. Taken from [7]

of flow cycles.

The increase in standard deviation was said to be almost linear with the increasing number of flow cycles. The authors incorporated this degradation in read accuracy into their sequencing simulator.

Unlike MetaSIM, Flowsim produces flowgrams in addition to sequencing reads. The output produced by Flowsim is similar to the output produced by a 454 sequencer as it is stored in the same SFF file format. SFF stands for Standard Flowgram Format. SFF files store the sequencing data produced by the 454 sequencing instruments.

The authors of Flowsim maintain the importance of producing flowgrams from sequencing simulation, in addition to sequencing reads. They state that the flow values in flowgrams contain important information that is not available in the pure nucleotide sequences (sequencing reads). They go on to describe several cases of sequencing analysis software that utilize flow values directly, such as PyroNoise by Quince et. al.[15]. PyroNoise utilizes a maximum likelihood algorithm to analyze flow values in order to establish whether a set of flowgram data is generated from one or several diverse genetic sequences.

Balzer et. al. refer to working with flow values (flowgrams) directly as working in flow-space, whereas working with sequencing reads is referred to as working in sequence-space.

3.2 MPP Read Alignment and Error Correction

In literature we see that there are two alternative environments for processing MPP data, flow-space and sequence-space. Both MPP read alignment and error correction algorithms can be classified into sequence-space or flow-space algorithms depending on their modus operandi.

We have established sequence-space to involve working with sequencing reads. We

have further established that a sequencing read is the product of performing base calling on a flowgram. However, we also know that base calling is an inaccurate procedure, due to flow value ambiguity. According to Vacic et. al. [18], during base calling, flow value ambiguity leads to the loss of valuable sequencing information.

Therefore the sequencing reads input into a sequence-space algorithm are already potentially inaccurate. A sequence-space algorithm may therefore produce a flawed result, irrespective of how well the algorithm itself attempts to solve the problem.

This argument is used in literature to advocate for flow-space algorithms as a more accurate alternative to sequence-space algorithms.

The following subsections are organized as follows:

Subsection 3.2.1 discusses sequence-space and flow-space algorithms for MPP read alignment.

Subsection 3.2.2 discusses sequence-space and flow-space algorithms for MPP read error correction.

3.2.1 MPP Read Alignment

We have established that in an MPP experiment, there is a reference genome present. The reference genome is from the same homologous genomic region being sequenced. The presence of a reference genome facilitates the determination of the correct context of each MPP read.

We have further established that MPP read alignment may be conducted in flow-space or sequence-space.

Subsection 3.2.1.1 discusses MPP read alignment algorithms designed to work in sequence-space.

Subsection 3.2.1.2 discusses MPP read alignment algorithms designed to work in flow-space.

3.2.1.1 Sequence-space alignment

Sequence-space alignment refers to the alignment of the sequencing reads generated in an MPP experiment with respect to a reference genome nucleotide sequence.

[21] provides a list of proprietary and open source sequence-space short read alignment software. However some software listed here is specifically meant for aligning short reads generated by other next generation sequencing methodologies apart from 454 sequencing.

The software packages mentioned here that do cater to 454 sequencing mostly implement local pairwise alignment schemes for the individual pairwise alignment of each read to a reference genome. Some algorithms further attempt to pairwise align each individual read to each nucleotide sequence in a database of reference genomes.

The key difference in the short read alignment software algorithms listed in [21] is differing approaches used to index the reference genome or the database of reference genomes as an optimization for the individual local pairwise alignment of reads.

In literature, we identified four distinct sequence-space alignment algorithms.

(i) Smith-Waterman alignment

(ii) Asymmetric Smith-Waterman alignment

(iii) Gap Propagation

(iv) PyroAlign

The Smith-Waterman and Asymmetric Smith-Waterman alignment algorithms implement individual local pairwise alignment of each sequencing read to a reference genome sequence.

Gap Propagation and PyroAlign go a step further to align the list of sequencing reads generated from an MPP experiment with respect to each other in addition to each read being locally aligned with respect to the reference genome sequence.

We provide a description of each of the algorithms below. We also describe the complexity of each algorithm.

In our description of complexity in this subsection, we assume a total of $\boldsymbol{N}$ sequencing reads with an average read length of L_R and a reference genome, Γ, of length L_Γ.

I) Smith-Waterman alignment

According to Eriksson et. al. [3], a simple pairwise local alignment of a sequencing read with the reference genome suffices in determining the correct context of the read. This involves the use of a local alignment algorithm, such as the Smith-Waterman algorithm (SW), to align the read to the reference genome. An example that typifies the local alignment of a sequencing read with the reference genome alignment is shown in Figure 3.4.

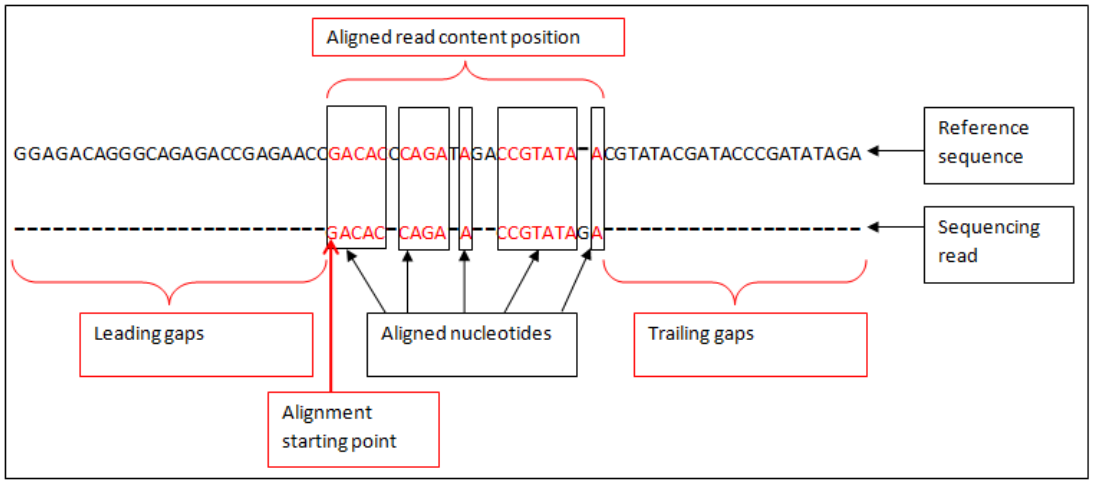

Figure 3.4: Pairwise local alignment of a sequencing read with the reference genome - The alignment requires the insertion of leading and trailing gaps into the sequencing read

The original SW algorithm for local alignment, described in [20], is characterized by Equation 3.1.

Equation 3.1:

$$\text{V(i,j)} = \max \begin{cases} 0 \\ \text{V(i-1,j-1)} + \text{score}(S_i, S_j) \\ \max\limits_{1 \leq k \leq i} \{\text{V(i-k,j)} + W_k\} \\ \max\limits_{1 \leq l \leq j} \{\text{V(i,j-l)} + W_l\} \end{cases}$$

- V(i,j) - Score of the alignment of subsequence of reference genome ($S_{1...i}$) with subsequence of sequencing read ($S_{1...j}$)
- W_k - Penalty incurred when inserting a gap of length k into the sequencing read. W_k = -1 + ($-\frac{1}{3} \times k$). (Where -1 is the gap open penalty and $-\frac{1}{3}$ is the gap extension penalty).
- W_l - Penalty incurred when inserting a gap of length l into the reference sequence. W_l = -1 + ($-\frac{1}{3} \times l$)

Equation 3.2 is a generalization of the original SW algorithm [22].

Equation 3.2:

$$\text{V(i,j)} = \max \begin{cases} 0 \\ \text{V(i-1,j-1)} + \text{score}(S_i, S_j) \\ \text{V(i-1,j)} + \text{score}(S_i, -) \\ \text{V(i,j-1)} + \text{score}(-, S_j) \end{cases}$$

According to Saeed et. al. [4], the time complexity of simple pairwise alignment of each read in a set of sequencing reads with the reference genome, is $\Theta(N \times L_R \times L_\Gamma)$.

II) Asymmetric Smith-Waterman alignment

Equation 3.3:

$$\text{V(i,j)} = \max \begin{cases} 0 \\ \text{V(i-1,j-1)} + \text{score}(S_i, S_j).\boldsymbol{w} \\ \text{V(i-1,j)} + \text{score}(S_i, -) \\ \text{V(i,j-1)} + \text{score}(-, S_j).\boldsymbol{w} \end{cases}$$

- V(i,j) - Score of the alignment of subsequence of reference genome ($S_{1...i}$) with subsequence of sequencing read ($S_{1...j}$)
- $\boldsymbol{w}$ - Weight given in Equation 3.4 below

Alignment parameters: match = 1, mismatch = -3, gap open penalty = -5, gap extension penalty = -2

Wang et. al. perceived that in order to facilitate more accurate local alignment of a sequencing read with the reference genome, a modification the SW alignment algorithm is necessary [13]. The authors modify the SW algorithm by incorporating the phred-equivalent quality score for each nucleotide within the sequencing read. We previously established the formula for the phred-equivalent quality score of a nucleotide within a sequencing read in Equation 2.2.

The modified local alignment algorithm described by Wang et. al. is called the Asymmetric Smith Waterman (ASW) algorithm. ASW is characterized by applying weights derived from the phred-equivalent quality scores to the sequencing read. These weights are applied to the sequencing read only and not to the reference sequence, hence the mention of asymmetry.

Given the generalization of the SW algorithm in Equation 3.2, the authors of ASW

characterize the ASW algorithm by Equation 3.3, where the parameters for alignment are explicitly specified.

Note that in applying the formula in Equation 3.3 for ASW, $\boldsymbol{i}$ is used to index nucleotide bases in the reference sequence and $\boldsymbol{j}$ is used to index nucleotide bases in the sequencing read. Given a nucleotide base S_j in the sequencing read, a weight $\boldsymbol{w}$ is applied to the alignment of S_j as shown in Equation 3.3. $\boldsymbol{w}$ is defined by Equation 3.4.

Equation 3.4:

For a nucleotide S_j in a sequencing read:

$$w = 1.0 \ - \ 10^{-\frac{q}{10}}$$

where $\boldsymbol{q}$ is the phred-equivalent quality score of S_j in the read (obtained by applying Equation 2.2).

Note that ASW has the same time complexity as SW.

III) Gap Propagation

We have established that the two solutions we have described (SW and ASW), align each individual sequencing read to the reference genome, but do not align the sequencing reads with respect to each other.

A full underlying genome sequence is estimated from sequencing read data by piecing sequencing reads (that originate from the specific underlying sequence) together. Therefore the sequencing reads must be aligned correctly with respect to each other in addition to being aligned correctly with respect to the reference genome.

Gap Propagation is described by Saeed et. al. in [4].

Gap Propagation is accomplished by aligning each read with the reference genome such that for each gap inserted into the reference genome, a gap is inserted in the same

position for all reads that overlap the genome in that position.

According to Saeed et. al. the time complexity of Gap Propagation is $\Theta(N^2)$.

IV) PyroAlign

Saeed et. al. [4] state that a perfect multiple alignment system for pyrosequencing reads is one in which:

1. The reads are aligned to each other in a way that conserves each individual reads position with respect to the reference genome sequence
2. The reads have maximum overlap
3. The aligned reads all have the same length (which includes leading and trailing gaps) in the final alignment.

To this end Saeed et. al. developed a novel algorithm for the multiple alignment of pyrosequencing reads, which aligns sequencing reads relative to each other as well as the reference genome. The algorithm, PyroAlign, involves two fundamental steps:

1. The Semi-Global (SG) alignment of each sequencing read to the reference sequence. This alignment positions each sequencing read into the correct context. SG alignment is a modification of the standard Needleman-Wunsch algorithm for global alignment. It is useful for aligning a sequencing read to the reference sequence because it does not penalize the leading and trailing gaps in the sequencing read. SG alignment is exactly like Needleman-Wunsch algorithm, except the first row and column of the dynamic programming matrix are initialized to 0 (in order to not penalize leading gaps) and the optimal alignment is detected as the maximum value on the last row or column (in order not to penalize trailing gaps).
2. The hierarchial progressive alignment of the sequencing reads with respect to each other, in order to form a multiple alignment of sequencing reads. This step in

turn involves two sub-steps:

a) The reads are reordered to generate a guidance tree for the alignment. This is such that reads that are closer in distance from each other border each other more closely in the guidance tree. The authors exploit the fact that the reads are coming from highly similar sequences and therefore reads that are aligned near the same starting point with respect to the reference genome are likely to be similar.

   Let us take N reads $R_1, R_2, ..., R_N$ generated from MPP. Further let us assume that L_Γ represents the length of the reference sequence and $L(R)_p$ is a generic representation for the length of a sequencing read. Then a sequencing read that has been SG aligned with the reference sequence can be represented by R_{pq}, where the p^{th} read has $\boldsymbol{q}$ leading gaps and $L_\Gamma - q - L(R)_p$ trailing gaps. We reorder the reads such that R_{pq} precedes $R_{p(q+1)}$ $\forall p, q \in L_\Gamma$.

b) This sub-step is conducted by:

   - Performing pairwise local alignment using standard Needleman-Wunsch on each pair of neighboring aligned sequencing reads using the ordering described in the previous sub-step. This results in $\frac{N}{2}$ pairs of aligned reads.
   - Constructing a binary tree for hierarchial profile-profile alignment. This is shown in Figure 3.5 taken from [4]. The tree is constructed according to the ordering of the reads obtained in sub-step (a).
   - Performing a multiple alignment of the aligned reads obtained according to the constructed binary tree.

According to Saeed et. al., the time complexity of PyroAlign is $\Theta(NlogN \times L_\Gamma^2 + NL_R^2)$.

In the experiments performed by Saeed et. al., simple SW/ASW alignment of each

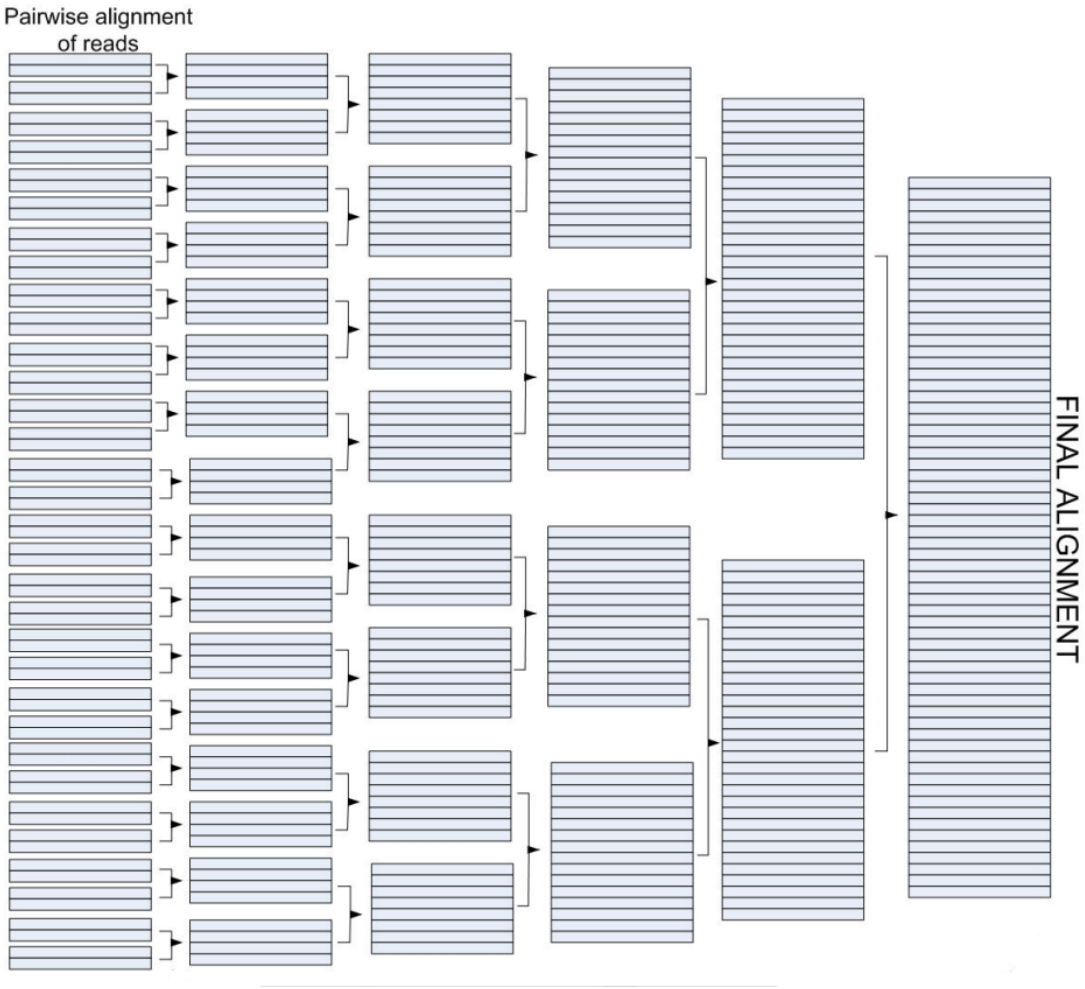

Figure 3.5: Saeed et. al. binary tree for hierarchial profile-profile alignment - The $\frac{N}{2}$ pairs of sequencing reads are the starting point for profile-profile alignment [4]

read with the reference genome produces an overall read alignment quality that is inferior to both PyroAlign and Gap Propagation. We recall that this is because of the established fact that SW/ASW alignments align each sequencing read with respect to the reference genome and do not align sequencing reads with respect to each other.

Given an MPP experiment that produces several sequencing reads, we only consider alignment solutions produced by PyroAlign and Gap Propagation as complete multiple alignments of the sequencing reads.

Saeed et. al. further claim that PyroAlign consistently outperforms Gap Propagation in terms of overall alignment accuracy [4].

3.2.1.2 Flow-space alignment

Flow-space alignment involves aligning the raw flowgrams (flow value progressions) obtained from MPP with a flow-space representation of the reference genome.

Specifically, flow-space alignment seeks to align each MPP flowgram in the possible position that it originated from with respect to the flow space representation of the reference genome.

If we take Γ to represent the reference genome, a flow-space representation of Γ is obtained by:

1. Deriving a run-length encoding of Γ
2. Inserting zero length homopolymer specifications into the result of 1. to obtain a run-length encoding that adheres to the contiguous cyclical flow of nucleotides in the T-A-C-G order characteristic of MPP.

We have identified two distinct flow-space alignment algorithms in literature:

(i) Quince et. al. flow-space alignment

(ii) Flowgram Matching

We describe each algorithm and state the algorithm complexities. For this subsection, we will refer to the flow-space representation of the reference genome as $\boldsymbol{G}$.

I) Quince flow-space alignment

Quince et. al. state that in order to align a flowgram, $\boldsymbol{f}$, of length $\boldsymbol{M}$, with respect to $\boldsymbol{G}$, we should find the segment in $\boldsymbol{G}$ that maximizes the likelihood of the progression of flow values in $\boldsymbol{f}$.

Given a segment, $\boldsymbol{U}$, of $\boldsymbol{G}$, the likelihood of $\boldsymbol{f}$ is summarized by Equation 3.5.

Equation 3.5:

$$P(f \sim U) \;=\; \prod_{l=1}^{M} P(f^l | n = U^l)$$

- $P(f \sim U)$ - Probability that $\boldsymbol{f}$ matches $\boldsymbol{U}$
- f^l - Flow value at position $\boldsymbol{l}$ in $\boldsymbol{f}$
- U^l - Homopolymer length at position $\boldsymbol{l}$ in $\boldsymbol{U}$
- $P(f^l | n = U^l)$ - The likelihood of flow value f^l given homopolymer length U^l. This is obtained from the FVDM being used to model MPP

The authors therefore seek a segment $\boldsymbol{U}$ in $\boldsymbol{G}$ for which $P(f \sim U)$ in Equation 3.5 is maximized. They however claim that these probabilities are very small and opt to work with the negative logarithm of $P(f|U)$.

According to Quince et. al., the distance, $\boldsymbol{d}$, between a flowgram $\boldsymbol{f}$ and a segment $\boldsymbol{U}$ of $\boldsymbol{G}$ is given by Equation 3.6. $\boldsymbol{f}$ is therefore aligned to the segment, $\boldsymbol{U}$ of $\boldsymbol{G}$, for which $d(f, U)$ is minimized.

Equation 3.6:

$$
\begin{aligned}
d(f,U) &= -log(\prod_{l=1}^{M} P(f^l|n=U^l)) \\
&= \sum_{l=1}^{M} -logP(f^l|n=U^l) \\
&= \sum_{l=1}^{M} d(f^l,U^l)
\end{aligned}
$$

- $d(f^l,U^l)$ - Distance between the flow value, f^l, at position $\boldsymbol{l}$ in $\boldsymbol{f}$ and the homopolymer length U^l at position $\boldsymbol{l}$ in $\boldsymbol{U}$.

A brute-force approach to determining the correct alignment of a flowgram, $\boldsymbol{f}$ of length $\boldsymbol{M}$, with respect to $\boldsymbol{G}$ would be to align $\boldsymbol{f}$ with all $\boldsymbol{M}$ long segments in G and report the segments that produce the best alignments (segments for which $d(f,U)$ is minimized). According to Vacic et. al. [18], this algorithm runs in prohibitive computational time, because an MPP experiment produces a large volume of flowgrams, each of which would have to be aligned with respect to a possibly long reference genome.

Quince et. al. therefore develop an optimization to the alignment of $\boldsymbol{f}$ with respect to $\boldsymbol{G}$. The authors use an adaptation of the global Needleman-Wunsch (NW) alignment algorithm.

In this adaptation, the cost of matching a flow of intensity f^l to a homopolymer of length $\boldsymbol{n}$ is given by Equation 3.7.

Equation 3.7:

$$d(f^l, n) = -log(P(f^l|n))$$

- $d(f^l, n)$ - Cost of matching a flow value of intensity f^l to a homopolymer of length $\boldsymbol{n}$.

The gap cost is set at a value just larger than $d(1.0|0)$, so that introducing a gap is preferred to matching a signal to no signal. Gaps must be introduced in gaps of four to emulate the period of nucleotide flows in MPP. The cost of the gaps is included into the measure of the distance between $\boldsymbol{f}$ and $\boldsymbol{U}$.

The authors state that it is useful to normalize $d(f, U)$ by the alignment length $\boldsymbol{M}$ so that short flowgrams are not considered to be more likely to have originated from the same sequence than a longer flowgrams. Therefore the normalized score of a flowgram alignment is given by Equation 3.8.

Equation 3.8:

$$d'(f, U) = \frac{d(f,U)}{M}$$

- $d'(f, U)$ - Normalized score for the alignment of $\boldsymbol{f}$ with $\boldsymbol{U}$

We establish the complexity of the Quince et. al. approach to aligning each flowgram of length $\boldsymbol{M}$ in a list of $\boldsymbol{N}$ quality-trimmed flowgrams to a flow-space representation of the reference genome based on two assumptions:

1. The length of the flow-space representation ($\boldsymbol{G}$) of a reference genome (Γ) is considered to be roughly proportional to the length of the underlying reference genome i.e. $L_G \propto L_\Gamma$, where L_G and L_Γ are the length of the flow-space representation of the reference genome and the length of the reference genome respectively.

2. The length of a quality-trimmed flowgram generated from a fragment $\boldsymbol{R}$ of length L_R is roughly proportional to L_R i.e. $L_F \propto L_R$, where L_F is the length of the flowgram.

We surmise that the complexity of pairwise aligning $\boldsymbol{N}$ flowgrams is approximately $\Theta(N \times L_R \times L_\Gamma)$.

II) Flowgram Matching

Vacic et. al. describe their approach to flow-space alignment in [18]. The authors call their algorithm Flowgram Matching.

The theory behind Flowgram Matching is very similar to the Quince et. al. approach to flow-space alignment. The key difference between these two approaches is that Flowgram Matching aligns a flowgram $\boldsymbol{f}$ to a segment $\boldsymbol{U}$ of $\boldsymbol{G}$, based on the posterior probabilities of the progression of homopolymer lengths in $\boldsymbol{U}$ given the progression of flow values in $\boldsymbol{f}$. Recall that Quince et. al. align $\boldsymbol{f}$ to $\boldsymbol{U}$ based on the likelihood of the flow value progression in $\boldsymbol{f}$ given the homopolymer progression in $\boldsymbol{U}$.

Let us make this distinction clear. Let us assume $\boldsymbol{f}$ is an MPP flowgram of length $\boldsymbol{M}$. According to the theory behind Flowgram Matching, the probability that $\boldsymbol{F}$ matches a segment $\boldsymbol{U}$ of $\boldsymbol{G}$ is can be stated, using the terminology of Quince et. al., as Equation 3.9.

A comparison of Equation 3.9 to Equation 3.5 establishes that the former bases the matching of $\boldsymbol{f}$ to $\boldsymbol{U}$ on posterior probability whereas the latter bases the same on likelihood.

Recall from Bayesian statistics and from Equation 2.1, that a posterior probability is obtained from likelihoods and prior probabilities.

Based on this, we cannot necessarily establish one of these approaches to flow-space alignment to be better than the other. Flowgram Matching may only be considered to

be superior to the Quince et. al. approach if the prior probabilities for homopolymer lengths are very accurate.

Equation 3.9:

$$P(f \sim U) = \prod_{l=1}^{M} P(n = U^l | f^l)$$

- $P(f \sim U)$ - Probability that $\boldsymbol{f}$ matches $\boldsymbol{U}$
- f^l - Flow value at position $\boldsymbol{l}$ in $\boldsymbol{f}$
- U^l - Homopolymer length at position $\boldsymbol{l}$ in $\boldsymbol{U}$
- $P(n = U^l | f^l)$ - The posterior probability of homopolymer length U^l given the flow value f^l

We now look at Flowgram Matching in detail.

Let us assume that $\boldsymbol{F}$ is a flowgram obtained from MPP such that the former can be represented as a sequence of $\boldsymbol{m}$ flows as shown: $\boldsymbol{F}$=$(b_0, f_0), (b_1, f_1), ..., (b_{m-1}, f_{m-1})$, where b_i is the nucleotide flowed and f_i is the flow signal observed as a result.

Then the probability that the flowgram, $\boldsymbol{F}$, matches a segment in $\boldsymbol{G}$ from position $\boldsymbol{k}$ is given by Equation 3.10. Note that Equation 3.10 is an alternative way of stating Equation 3.9.

Using the Bayesian relationship established in Equation 2.1 and assuming a null model where each homopolymer length probability is obtained by its frequency in $\boldsymbol{G}$, Flowgram Matching is cast as an analog of profile matching, where the scoring matrix is defined by Equation 3.11. For a complete derivation of this result, see [18].

Equation 3.10:

$$P(F \sim G_{k...k+m-1}) = \prod_{i=0}^{m-1} P_{b_i}(L = g_{k+i} | S = f_i)$$

- $\boldsymbol{L}$ - Random variable denoting the length of the homopolymer in $\boldsymbol{G}$.
- $\boldsymbol{S}$ - Random variable denoting the signal strength in $\boldsymbol{F}$ potentially induced by $\boldsymbol{L}$.
- g_{k+i} is the length of the homopolymer $\boldsymbol{i}$ positions down from the beginning of the match $\boldsymbol{k}$.
- $P_{b_i}(L = g_{k+i} | S = f_i)$ - The posterior probability of the homopolymer length $L = g_{k+i}$ given the flow value $S = f_i$.

Equation 3.11:

$$M_{i,j} = logP_{b_i}(S = f_i | L = j) \; - \; logP_{b_i}(S = f_i)$$

Equation 3.12:

$$Score(F \sim G_k) \; = \; \sum_{i=0}^{m-1} M_{i,j}$$

According to Vacic et. al., a flowgram is therefore aligned to the segment of the flow-space representation of the reference genome that maximizes the score defined in Equation 3.12.

We now discuss the Vacic et. al. optimization to the determination of this score.

Vacic et. al. begin by defining a threshold value, $\boldsymbol{t}$, for the score of aligning a flowgram, $\boldsymbol{f}$, to a segment of $\boldsymbol{G}$. This is such that only the segments of $\boldsymbol{G}$ that yield a score greater than or equal to $\boldsymbol{t}$ when aligned with $\boldsymbol{f}$ are considered to pinpoint a possible original alignment of $\boldsymbol{f}$ with respect to $\boldsymbol{G}$.

The authors then utilize lookahead scoring, whereby if the flow values in $\boldsymbol{f}$ are being aligned with the homopolymer lengths in a segment of $\boldsymbol{G}$, we can predict whether the overall score of the alignment will exceed the threshold score $\boldsymbol{t}$ based on the alignment score of each flow value with each homopolymer length. We can therefore terminate an alignment that will not exceed the threshold in advance. Lookahead scoring is described by Beckstette et. al. in [19].

A further optimization by Vacic et. al. is the indexing of the flow-space representation of the reference genome using an Enhanced Suffix Array (ESA). This enables simultaneous searches for suffixes (segments) of the reference genome that match the flowgram $\boldsymbol{f}$. ESAs are described in [19].

According to [19], this algorithm runs in sub-linear time.

3.2.2 MPP Read Error Correction

We define MPP read error correction as the determination of the authentic variation in the aligned MPP reads for each nucleotide/flow position covered by MPP in the genome.

The approaches to MPP read error correction can be classified into two categories:

1. Sequence-space error correction

2. Flow-space error correction

Subsections 3.2.2.1 and 3.2.2.2 discuss sequence-space and flow-space error correction respectively.

3.2.2.1 Sequence-space Error Correction

According to Wang et. al., in sequence-space, most of the errors in sequencing reads are singleton mismatch errors, where only a single sequencing read contains an insertion, deletion or substitution with respect to the original genome being sequenced [13].

Let z be a nucleotide position covered by an MPP experiment. Then for all the MPP reads aligned at position z, we perform sequence-space error correction by removing non-authentic nucleotide variation at position z (where non-authentic variation is variation resulting from sequencing error).

We discovered three distinct sequence-space error correction alignment algorithms in literature.

(i) Wang et. al. error correction

(ii) Eriksson et. al. error correction

(iii) Zagordi et. al. error correction

We discuss each of these distinct approaches to sequence-space error correction.

I) Wang et. al. error correction

The approach taken by Wang et. al. in [13] begins with the classification of homopolymeric regions (regions containing repeats of three or more identical bases and the flanking non-identical bases) and non-homopolymeric regions.

From an MPP sequencing experiment, Wang et. al. empirically observed that the distribution of sequencing errors in non-homopolymeric regions can be estimated by a Poisson distribution with a mean, $\mu = 0.0007$. They estimate the distribution of sequencing errors in homopolymeric regions by a Poisson distribution with a mean $\mu = 0.0044$.

The authors state that for a single nucleotide position, given a distinct variant nucleotide with $\boldsymbol{n}$ occurrences in $\boldsymbol{N}$ reads, the probability that the variant nucleotide occurs $\boldsymbol{n}$ or more times if it were caused by a sequencing error, is given by Equation 3.13.

Equation 3.13:

$$P = 1 - \sum_{k}^{n-1} \frac{e^{\lambda} \times \lambda^{k}}{k!}$$

- P - Probability that specific variant would occur **n** or more times if it were a sequencing error.
- μ - Mean error rate per nucleotide position (Recall $\mu = 0.0007$ in non-homopolymeric regions and $\mu = 0.0044$ in homopolymeric regions)
- λ - Expected number of errors. This is calculated by $\lambda = N \times \mu$

For a given nucleotide position, Wang et. al. consider variant nucleotides that yield a value for $\boldsymbol{P}$ in Equation 3.13 less than or equal to 0.001 $(P < 0.001)$ to be highly unlikely to be sequencing errors.

Let us take a critical look at this approach to sequence-space error correction.

Recall that, based on MPP experiments that provide conflicting FVDMs, we established that results obtained from an MPP experiment may be biased by systematic errors. Given this argument we question the extent to which empirically observed parameters can be used for accurate error correction in a generic sense.

Secondly, the view of error distributions as being organized into two categories, such that there is an error distribution for homopolymeric regions and an error distribution for non-homopolymeric regions, may be too much of a generalization. Perhaps a finer

categorization should be applied to error distributions (otherwise the error rate for a homopolymer of length 3 has the same distribution as the error rate for a homopolymer of length 9).

The authors do not provide a complexity analysis for this method of sequence-space error correction.

II) Eriksson et. al. error correction

The second approach to sequence-space error correction is described by Eriksson et. al. in [3]. The authors make use of a fact that we previously established in subsection 2.1.3, this being that a small proportion of the MPP reads in a sequencing experiment are responsible for the majority of the total errors in the data set. Eriksson et. al. therefore claim that in an MPP experiment the error rate is approximately 5 to 10 errors per kb, and discarding 10% of the MPP reads (the ones with ambiguous bases) reduces this error rate to 1 to 3 errors per kb.

Eriksson et. al. conduct MPP error correction by constructing windows over segments of a multiple alignment of MPP reads.

Given a window, the authors begin with a null hypothesis of a single genomic sequence in the window, such that any variation in the MPP reads is hypothesized to be due to a uniform MPP sequencing error ϵ. Subsequently the consensus nucleotide base in each column of the window is determined.

Note that under the null hypothesis, the cumulative count for the number of distinct genomic sequences in the window should be 1.

Following the declaration of the null hypothesis, Eriksson et. al. estimate the total number of distinct genomic sequences, $\boldsymbol{k}$, in the window by the steps outlined below:

(i) For each column of the window, each non-redundant nucleotide mutation (deviation from the consensus base) is tested for over-representation/significance. If the

number of variant nucleotides of this specific nucleotide type within the column exceeds the expectation under the null hypothesis, there is evidence for a distinct variant sequence (a genomic sequence different from the consensus).

Note:

- According to Eriksson et. al., given the null hypothesis and $\boldsymbol{d}$ reads that overlap in a window, the probability of observing $\boldsymbol{x}$ or more occurrences of a distinct variant nucleotide base within a single position of the input multiple alignment is given by the binomial distribution outlined in Equation 3.14.

  If the value of $Pr(X \geq x)$ calculated by Equation 3.14 is less than a given p-value, we reject the null hypothesis of a single genomic sequence in the window and increase the cumulative count of the number of distinct genomic sequences in the window by 1.

- The same principle is used in Equation 3.14 to calculate the significance of a mutation within a column (alignment position) of an input multiple alignment as in equation 3.13 for Wang error correction. The difference is in the nature of the estimated MPP error rate distributions.

  Whereas Wang et. al. model the MPP error rate distribution by a Poisson distribution, Eriksson et. al. model the same by a Binomial distribution. Further, similarly to Equation 3.13, Equation 3.14 is based on an empirically observed error rate. Therefore we also question the general applicability of Equation 3.14.

  Eriksson error correction, however, goes a step further because this significance calculation is only a small part of the algorithm.

  Further, the Eriksson calculation is localized within windows of the multiple alignment. This finer granularity should produce more accurate results than

the calculation conducted in the Wang algorithm, which does not divide the input multiple sequencing read alignment into windows. The Wang algorithm perceives each position of the multiple alignment in the context of the entire alignment.

Equation 3.14:

$$Pr(X \geq x) \;=\; \sum_{k=x}^{d} \binom{d}{k} \epsilon^{k} (1-\epsilon)^{d-k}$$

- ϵ - MPP error rate
- d - Number of reads that overlap in the window

(ii) For every two distinct alignment columns $\boldsymbol{u}$ and $\boldsymbol{v}$ in the window, each pair of mutations is tested for significance. If the number of co-occurrences of the mutation pair exceeds the expectation under the null hypothesis, there is evidence for a genomic sequence different from the consensus.

Note:

- According to Eriksson et. al., given the null hypothesis, the probability of observing $\boldsymbol{c}$ co-occurrences of a mutation pair is given by the hypergeometric distribution outlined in Equation 3.15 (according to Fisher's exact test).

  Given $\boldsymbol{c}$ co-occurrences of a mutation pair, if the value of $Pr(C = c)$ according to Equation 3.15 is less than a given p-value, we reject the null hypothesis of a single genomic sequence in the window and increase the cumulative count of the number of distinct genomic sequences in the window by 1.

Eriksson et. al. correct for any over-counting that results from the procedure detailed above.

Having established the total number of distinct genomic sequences ($\boldsymbol{k}$) for a window, the authors cluster all the sequencing reads that overlap in the window by using k-means clustering. Each read is then corrected to the consensus sequence of its cluster. The consensus sequence of a cluster is calculated from weighted counts based on the quality scores of the nucleotides in the sequencing reads.

Equation 3.15:

$$Pr(C = c) = \frac{\binom{n_v}{c}\binom{d-n_v}{n_u-c}}{\binom{d}{n_u}}$$

- d - Number of reads that overlap in the window
- n_u - Number of times specific mutation has been observed in column $\boldsymbol{u}$
- n_v - Number of times specific mutation has been observed in column $\boldsymbol{v}$

The authors use three collections of contiguous windows on a multiple alignment of sequencing reads. The windows are shifted relative to each other, such that each base is covered exactly three times. The concluding correction of a base is therefore the consensus from the three corrections on the base.

The authors do not provide a complexity analysis for this method of sequence-space error correction.

III) Zagordi et. al. error correction

Similarly to Eriksson et. al. error correction, Zagordi et. al. error correction is based on the principle of clustering sequencing reads.

In fact Zagordi et. al. error correction is based on the assumption that, given

a set of sequencing reads generated from an MPP experiment in which a mixture of variants is sequenced, the reads will tend to cluster around their corresponding variant sequences (i.e. the variant sequences from which they are derived) with a distribution that is directly influenced by the MPP error process. Further, the authors argue that the distinct variants are separated by the true evolutionary distances between them.

The authors state that the main limitation of Eriksson et. al. error correction is that the error rate utilized in Equation 3.14 has to be assumed.

Zagordi et. al. also recognize that given a set of sequencing reads generated from an MPP experiment, there is a high uncertainty as to the number of distinct variant sequences (i.e. the number of distinct clusters) underlying the reads.

Following this, the authors devise a solution for the clustering problem that utilizes a tool in Bayesian statistics called the Dirichlet Process Mixture Model (DPM), whereby the number of clusters is increased dynamically during the clustering of reads. The DPM is described in [28].

The algorithm in [2] begins with the random assignment of the sequencing reads to $\frac{n}{10}$ variant sequences, where n is the total number of sequencing reads. The variant sequences are randomly generated from a reference genome sequence.

Subsequently the cycle below, where model parameters such as the MPP sequencing error rate are continually recalculated, is repeated over and over again. Note that the repetition of this cycle is the application of a principle known as Gibbs sampling.

1. The probabilities of the variant sequences given the reads that have been assigned to them are recalculated.

2. The MPP error rate parameter that generates potentially errant sequencing reads from the variant sequences is recalculated.

3. The variant sequence parameter that generates variant sequences from the provided reference genome sequence is recalculated.

4. The sequencing reads are assigned to the existing variant sequences (clusters) or they instantiate a new cluster according to a formulated posterior distribution for the DPM.

Gibbs sampling for DPMs is discussed in detail in [29].

After a burn-in phase, the assignment of each read to the variant sequences is noted. If a read is assigned to a variant sequence for 80% of the repetitions of the above cycle, the read is considered to originate from that specific variant sequence.

In [2], the clustering procedure discussed here is conducted in successive windows along the genome. The final assignment of a read to a variant sequence (cluster) follows a majority rule.

Zagordi et. al. do not provide a complexity analysis for this method of sequence-space error correction.

In comparison with Wang and Eriksson error correction, we note that Zagordi error correction does not utilize empirically observed values. In theory, therefore, this method of error correction should be more applicable to MPP experiments in general as it does not contain any bias towards a particular set of MPP results. However, the trade-off is a relatively much longer execution time for the Zagordi algorithm, as the Gibbs sampler will require multiple (≥ 1000) iterations to converge to a solution [29].

3.2.2.2 Flow-space Error Correction

Given a number of flowgrams from an MPP experiment, flow-space error correction seeks to generate the set of authentic sequences that produced the flowgrams.

Similarly to the last two sequence-space algorithms discussed, this is often conducted by implementing a clustering algorithm, such that similar flowgrams which potentially originate from the same variant sequence are clustered together.

Clustering similar flowgrams together enables us to predict an underlying sequence for each cluster. An underlying sequence for a cluster is an approximation of the variant

sequence that produced the flowgrams in the cluster.

We discovered one method of flow-space error correction in literature, Quince et. al. error correction.

I) Quince et. al. error correction

Quince et. al. implement a method of flow-space error correction that utilizes an Expectation Maximization (EM) algorithm to cluster MPP flowgrams in [15].

EM algorithms require a known number of clusters prior to commencement [15]. Recall that ideally, a cluster represents a set of MPP flowgrams that originate from a single distinct variant sequence. Therefore, in this case, the Quince et. al. algorithm requires a prior specification of the number of distinct variant sequences underlying a set of MPP flowgrams.

Following this, as a preliminary, the authors estimate the number of distinct variant sequences underlying a set of MPP flowgrams by using a complete linkage clustering algorithm. Complete linkage clustering is discussed in [27].

For complete linkage clustering Quince et. al. derive the distance between two flowgrams f_1 and f_2 by taking negative logarithm of the result of Equation 3.16 and normalizing by the length of the alignment overlap between f_1 and f_2, M.

The clusters formed by the complete linkage clustering algorithm at a given threshold distance $\boldsymbol{c}$ are used as input to the EM.

The expectation step of the EM is a computation of the matrix that represents the mapping of flowgrams to clusters according to Equation 3.17.

Subsequently, the maximization step of the EM involves three sub-steps:

1. The generation of new underlying sequences $j = 1...L$ using aligned flowgrams. For each sequence j, the homopolymer length at position l is given by Equation 3.18. Note that each calculated underlying sequence corresponds to a distinct

cluster of the input flowgrams.

2. The generation of new relative frequencies for the newly generated underlying sequences. This is done according to Equation 3.19.

3. The alignment of the flowgrams to the newly generated underlying sequences. This leads to new distances $d'(f_i, U_j)$.

Equation 3.16:

$$P(H|f_1, f_2) = \prod_{l=1}^{M} \frac{\sum_{n=0}^{\infty} P(f_1^l|n)P(f_2^l|n)P(n)}{\left(\sum_{n=0}^{\infty} P(f_1^l|n)P(n)\right)\left(\sum_{n=0}^{\infty} P(f_2^l|n)P(n)\right)}$$

- f_1, f_2 - The pair of flowgrams that we want to determine the distance between
- H - The hypothesis that f_1 and f_2 are from the same sequence
- M - The alignment length
- $P(f_1^l|n)$ - The likelihood of the flow value at position l in flowgram 1 given a homopolymer length n.
- $P(f_2^l|n)$ - The likelihood of the flow value at position l in flowgram 2 given a homopolymer length n.
- $P(n)$ - The prior probability of homopolymer length n. The list of prior homopolymer probabilities is obtained from **Supplementary Table** 1 in [15]

Equation 3.17:

$$z'_{i,j} = \frac{\tau_j exp\left(-\frac{d'(f_i, S_j)}{\sigma}\right)}{\sum_{k=1}^{L} \tau_k exp\left(-\frac{d'(f_i, S_k)}{\sigma}\right)}$$

- $z'_{i,j}$ - The expectation of the mapping of flowgram i to sequence j.
- τ_j - The relative frequency of sequence S_j.
- $d'(f_i, S_j)$ - The normalized distance between an aligned flowgram f_i and a sequence S_j.
- σ - The general cluster size

Equation 3.18:

$$U_j^l = min_n \left(\sum_{i=1}^{N} z'_{i,j} d'(f_i^l, n)\right)$$

Equation 3.19:

$$\tau_j = \sum_{i=1}^{N} \frac{z'_{i,j}}{N}$$

The maximization step of the EM seeks to maximize the complete likelihood of the flowgrams given the mapping of the flowgrams to clusters.

The EM converges such that the MPP flowgrams are clustered, whereby each cluster is defined by an underlying sequence. Each flowgram within a cluster will be aligned with respect to the corresponding underlying sequence. The EM also returns the relative frequencies of the sequences underlying the distinct clusters.

Quince et. al. do not provide a complexity analysis for this method of flow-space error correction.

Note that this method of error correction requires the input of two empirically observed values, these being the cut-off distance for the preliminary complete linkage clustering algorithm and the fixed size that characterizes the clusters in the EM algorithm. Therefore we cannot claim that the Quince algorithm is more generally applicable than the Wang and Eriksson algorithms. The most generally applicable algorithm from this perspective is therefore the Zagordi error correction algorithm.

The Quince algorithm EM requires multiple (≥ 1000) iterations to converge [15]. This algorithm therefore has an execution time that is comparable with the Zagordi algorithm.

4

MPP Simulation - Methodology

We implemented an MPP simulator in a Wolfram Mathematica 8.0 package named PyroSim. PyroSim can be accessed in two ways:

1. Via the MPPSoft web graphical interface at the URL:

   `http://hammer.cs.ukzn.ac.za/~anisa/`

2. Via the Mathematica 8.0 package, PyroSim, which can be downloaded at the URL:

   `http://hammer.cs.ukzn.ac.za/~anisa/PyroSim.html`

   The complete PyroSim user manual is available at the URL:

   `http://hammer.cs.ukzn.ac.za/~anisa/PyroSimManual.pdf`

We previously discussed the theory of MPP simulation in detail in subsection 3.1 of this thesis.

In our implementation of MPP simulation, we expanded the approach taken by Balzer et. al. [7]. Our sequencing simulation algorithm consists of 6 stages. A brief summary of each stage is provided below.

1. **Stage 1**: Our simulator PyroSim reads in a list (population) of sequences as input. Subsequently each sequence in the list is fragmented into sub-sequences. This results in a pool of fragments derived from the input population of sequences.

2. **Stage 2**: A run length encoding is derived from each fragment/sub-sequence obtained in stage 1. The run length encoding obeys the cyclical flow of nucleotide bases in MPP (in the order T-A-C-G for each flow cycle). Each run length encoding is known as a homopolymer progression.

3. **Stage 3**: Each homopolymer progression derived in the previous step is converted into a progression of flow values (i.e. a flowgram).

4. **Stage 4**: This is the flowgram refinement stage. This stage in itself is made up of two steps:

   - Given a flowgram, ambiguous flow cycles within the flowgram are identified. According to Huse et. al. [1], an ambiguous flow cycle is a cycle in which for all of the bases (T, A, C and G) flowed within the cycle, no significant flow value (light intensity) is observed. Following the identification of ambiguous flow cycles, the flowgram is trimmed so that it does not contain any ambiguous flow cycles.

     N.B.: This step is optional. A user utilizing our simulation package has the option whether or not to identify and trim ambiguous flow cycles.

   - Here a flowgram is quality trimmed. We discussed quality trimming in subsection 2.1.5.1 of this thesis. Our simulator quality trims each flowgram such that only a small proportion of flow values within the flowgram lie in the interval of overlap between a negative flow distribution (homopolymer length 0) and the distribution for a unit base incorporation (homopolymer length 1). These distributions are according to the FVDM being used for MPP simulation.

5. **Stage 5**: This stage of simulation involves outputting a flowgram graph for each flowgram.

6. **Stage 6**: The final stage of simulation involves translating each refined flowgram back into a sequence of nucleotides by base calling. Each nucleotide sequence derived from base calling is outputted along with a series of quality scores that represent the accuracy of each nucleotide within the nucleotide sequence.

Figure 4.1 is a flowchart of the stages of simulation implemented in our simulator. PyroSim also implements a module for sequencing simulation preliminaries. This module enables a user to generate the input population for sequencing simulation from an original DNA sequence.

In the following sections we discuss each component/stage of MPP simulation. Subsequently, we conclude with a description of how PyroSim links the components to achieve a realistic MPP simulation. We then provide a brief comparison of our simulator PyroSim to the simulator implemented by Balzer et. al. in [7], Flowsim.

Note: In each section we state the complexity of the PyroSim module discussed. We subsequently derive the overall computational complexity of PyroSim. The complexities are stated in terms of some of PyroSim's input parameters, these being:

- $\boldsymbol{N}$ - the size of the population of DNA nucleotide sequences input into PyroSim

- $\boldsymbol{L}$ - the mean length of a nucleotide sequence in the population of DNA nucleotide sequences input into PyroSim

- $\boldsymbol{l}$ - the mean length of a PyroSim DNA clone fragment

- $\boldsymbol{c}$ - the number of flow cycles set for sequencing simulation.

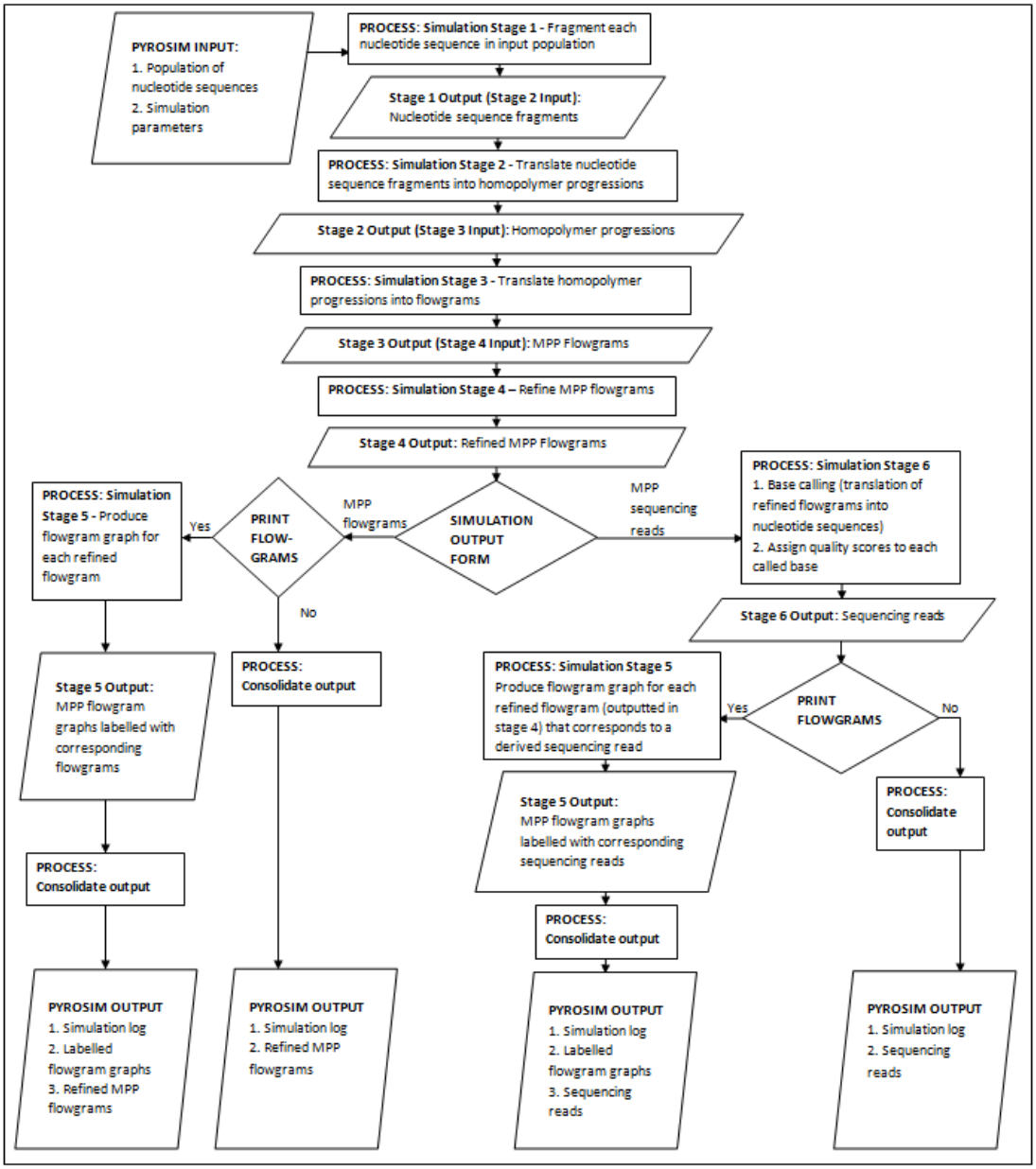

Figure 4.1: Flowchart of the stages comprising PyroSim MPP simulation algorithm - The algorithm is comprised by 6 distinct stages

4.1 Sequencing Simulation Preliminaries

4.1.1 Requirements and Specifications

We decided to provide the PyroSim user with the ability to generate a population of sequences that serves as the input for sequencing simulation. This provides the user with an alternative to directly supplying an existing population of input sequences.

To this end our simulator incorporates a module that generates a population of sequences according to the user's specifications. We call this module is called the ***generatePopulation*** module.

As input to the module, the user supplies an original sequence and explicitly specifies **variants** of the original sequence that should make up the desired output population.

A variant specification consists of two parts:

1. The mutation distance (as a percentage) from the original sequence
2. The number of copies to include in the output population

- **Note**: For simplicity we derive variants from the original user supplied sequence by substitutions only (and not by insertions or deletions of nucleotides). To this end, we define the mutation distance between a sequence and a derived variant as the percentage of substituted nucleotide bases in the variant with respect to the original sequence.

As a summary:

- ***generatePopulation* Module Input**: A user-specified original sequence and list of specifications for each distinct variant that the output population should contain.
- ***generatePopulation* Module Processing**: Each specified variant is generated by repeatedly mutating the original sequence until the best approximation of the

mutation distance pertaining to the variant is reached. The variant is then copied so that the number of copies of the variant in the output population matches the user's specification.

- ***generatePopulation* Module Output**: A population of nucleotide sequences.

4.1.2 Algorithms and Data Structures

In general, an invocation of the ***generatePopulation*** module should take the form:

$$\textbf{\textit{generatePopulation}}[\textbf{\textit{originalSequence}},\textbf{\textit{variantList}}]$$

where:

- ***originalSequence*** is a variable storing an original sequence from which the user wishes to generate genetic variants
- ***variantList*** is a list of the specifications of the variants required. ***variantList*** is represented as a $\boldsymbol{n}\times 2$ Mathematica List data structure where $\boldsymbol{n}$ is the number of variants specified in the list, each variant specification comprising of two numbers:
  1. d - A real number representing the mutation distance of the variant from ***originalSequence***, where $0 \leq d \leq 1$
  2. q - An integer representing the required number of copies of the variant, where $q \geq 1$

As an example, a typical invocation of the generate population method are shown below:

```
population=generatePopulation[originalSequence,{{0.03,5},{0.05,10},{0.07,10}}]
```

The above example invocation stores the result of the ***generatePopulation*** module in the variable ***population***.

The example creates three variants of ***originalSequence***. The first variant will be at a genetic distance of 0.03, the second at a distance of 0.05 and the third at a distance of 0.07 from ***originalSequence***. There will be 5, 10 and 10 copies of the first, second and third variants respectively in the outputted population.

The ***generatePopulation*** module makes use of three helper methods which are used in combination to generate variants of a user-specified original sequence:

(i) `mutate[seq_,pos_]`

(ii) `variant[seq_,dist_]`

(iii) `generateVariant[seq_,dist_,numVariants_]`

The full code for the generatePopulation module can be found in Appendix A.1 of this thesis.

The overall complexity of the ***generatePopulation*** module is $\Theta(NL)$.

4.2 Simulation Stage 1: Sample preparation

4.2.1 Requirements and Specifications

MPP involves a sample preparation stage prior to sequencing. This stage involves the fragmentation of the DNA being sequenced. Marguiles et. al. describe this fragmentation of the sample DNA sequences as being random [6].

Given a list (population) of nucleotide sequences as input, our simulator should be able to fragment each of the sequences, resulting in a pool of template DNA fragments. Although we have established this fragmentation to be random, PyroSim allows the user to specify the lengths of the fragments. The fragment lengths are specified according to distributions. Two options are available for this specification. The user can set the fragment lengths to follow a normal distribution and further assign the mean and

standard deviation for this distribution. Alternatively the user can specify the fragment lengths as following a uniform distribution with user-specified minimum and maximum parameters.

We incorporate a ***fragmentSequence*** module in PyroSim whose role is to cut each sequence in the list of input sequences into fragments. These fragments should have lengths that respect the normal or uniform user specified distribution for the fragment lengths. Obtaining a pool of fragments from a whole population or list of input sequences entails mapping the ***fragmentSequence*** module onto each individual sequence in the list of input sequences.

As a summary:

- ***fragmentSequence* Module Input**: An individual DNA sequence and a distribution specification, which determines how the input sequence will be fragmented.
- ***fragmentSequence* Module Processing**: The input sequence is cut into fragments whose lengths are distributed according to the distribution specification that is read as a parameter.
- ***fragmentSequence* Module Output**: A pool of DNA sequence fragments derived from the input DNA sequence.

4.2.2 Algorithms and Data Structures

The Mathematica code for the ***fragmentSequence*** module is provided in appendix A.2 of this thesis.

`fragmentSequence[seq_,dist_]` returns a list of fragments of an input sequence ***seq*** whose lengths comply to the distribution ***dist***. Each returned fragment is accompanied by a {start,end} positional vector indicating the position of the fragment in ***seq***.

The complexity of the ***fragmentSequence*** module is $\Theta(\frac{L}{l})$, where L and l are the length of ***seq*** and the mean of ***dist*** respectively.

A mapping of the module onto each nucleotide sequence in a population of $\boldsymbol{N}$ sequences results in an overall complexity of $\Theta(\frac{NL}{l})$ for stage 1 of sequencing simulation.

4.3 Simulation Stage 2: Translation of fragments into homopolymer progressions

4.3.1 Requirements and Specifications

This stage involves the translation of each individual fragment obtained from the sample preparation stage into a homopolymer progression.

Let us take the fragment **AAAGCCTAATTCCC** as an example. This fragment is converted into the desired homopolymer progression in two sub-stages.

1. **Sub-stage 1**: Tallying contiguous identical bases - Contiguous identical bases define a homopolymer.

   Taking our example fragment and representing it as a homopolymer progression, we end up with:

   {{A,3},{G,1},{C,2},{T,1},{A,2},{T,2},{C,3}}

2. **Sub-stage 2**: Expanding the homopolymer progression derived in Sub-stage 1 into a homopolymer progression that obeys the contiguous cyclical flow of nucleotides in the sequence T-A-C-G characteristic of MPP. This is done by inserting homopolymers of length 0 at appropriate positions. The length of the homopolymer progression derived in this substage is constrained by the number of flow cycles that are set for MPP simulation (Specifically the length of the homopolymer progression will be four times the number of MPP simulation flow cycles, because each flow cycle is constituted by T,A,C and G).

If the length of the obtained homopolymer progression exceeds four times the number of flow cycles, the excess flows are chopped off and only the subpart of the progression of length four times the number of flow cycles is returned.

On the other hand if the length of the obtained homopolymer progression is less than four times the number of flow cycles, homopolymers of length 0 are added onto the progression until the length of the homopolymer progression is equal to the required length.

Going back to our example, inserting homopolymers of length 0 in appropriate positions results in the homopolymer progression:
{{T,0},{A,3},{C,0},{G,1},{T,0},{A,0},{C,2},
{G,0},{T,1},{A,2},{C,0},{G,0},{T,2},{A,0},
{C,3},{G,0}},
which still translates to the original nucleotide sequence **AAAGCCTAATTCCC**.
The homopolymer progression derived in this sub-stage represents an ideal flowgram. This progression is the sequence of flow values that would be observed during the sequencing of the original nucleotide sequence if MPP were a completely error-free technology.

We implemented a ***sequencingReaction*** module in our simulator that carries out the processing described above.

The ***sequencingReaction*** module transforms an individual nucleotide sequence fragment into a progression of homopolymers. Therefore in order for the ***sequencingReaction*** module to transform an entire pool of fragments into a list of homopolymer progressions, the module must be mapped onto each fragment in the pool.

As a summary:

- ***sequencingReaction* Module Input**: A template DNA fragment and the num-

ber of flow cycles being run in the simulation

- ***sequencingReaction* Module Processing**: The template DNA fragment is translated into a homopolymer progression. The homopolymer progression is subsequently expanded to obey the contiguous cycling of DNA nucleotides in the order T-A-C-G for the specified number of flow cycles.

- ***sequencingReaction* Module Output**: A homopolymer progression of length 4 times the number of specified flow cycles.

4.3.2 Algorithms and Data Structures

We implemented both sub-stages of simulation stage 2 in a single algorithm.

`sequencingReaction[frag_,cycles_]` reads in a DNA fragment, ***frag***, and the number of flow cycles required for MPP simulation, ***cycles***. The module returns a homopolymer progression.

An outputted homopolymer progression is represented as a $\boldsymbol{n}\times 2$ Mathematica List of homopolymer specifications where $\boldsymbol{n}$ is the number of distinct homopolymers in the progression and each homopolymer is specified by:

1. A nucleotide base

2. The number of contiguous occurrences of the nucleotide base in the original fragment

The Mathematica code for the ***sequencingReaction*** module is provided in appendix A.3 of this thesis.

The complexity of the ***sequencingReaction*** module is $\Theta(c)$, where $\boldsymbol{c}$ is the required number of flow cycles.

If the ***fragmentSequence*** module in simulation stage 1 is mapped onto $\boldsymbol{N}$ nucleotide sequences, each of length $\boldsymbol{L}$ to obtain fragments of mean length $\boldsymbol{l}$, the total

number of fragments outputted by the ***fragmentSequence*** module will be $\frac{NL}{l}$. Subsequently if the ***sequencingReaction*** module is mapped onto each fragment derived in simulation stage 1, we obtain an overall complexity of $\Theta(\frac{NLc}{l})$ for simulation stage 2.

4.4 Simulation Stage 3: Translating homopolymer progressions into flowgrams

4.4.1 Requirements and Specifications

We mentioned that the progression of homopolymers obtained from the previous step represents an ideal sequence of flow values. This progression therefore needs to be modified to represent flow values that could be obtained from the sequencing instrument in reality.

In subsection 2.1.3, we established that the flow values that correspond to a homopolymer length $\boldsymbol{r}$ are distributed about $\boldsymbol{r}$ according to a FVDM.

PyroSim reads in the FVDM that it uses for MPP simulation from the user as a parameter. We opted for this rather than a fixed FVDM because of the contention over the true distribution of flow values about homopolymer lengths in literature [6, 7, 16, 17].

There is however a restriction on the user-supplied FVDM. The restriction is that positive flow distributions (distributions for homopolymer lengths greater than or equal to one) must be Gaussian (therefore the user specifies the mean and standard deviation for each distinct Gaussian positive flow distribution). The reason for this restriction will be provided later. The negative flow value distribution may take any shape or form. The user specifies the distributions for homopolymer lengths ranging from 0 to 9 in the input FVDM.

A PyroSim user has the option to use the default FVDM, which is version 2 of the

Richter et. al. model.

Our simulator implements a ***flowgram*** module that converts each homopolymer length in a homopolymer progression into a flow value according to the user-supplied FVDM.

In subsection 3.1.3 of this thesis we mentioned that the authors of Flowsim observed a degradation in the accuracy of flow values as one travels along a flowgram. Recall that we attributed this degradation to an increase in the standard deviations of the homopolymer distributions with increasing flow cycle number. PyroSim allows the user to set the gradient for this increase in standard deviations and further allows the user to set the number of interim flow cycles preceding each subsequent increase in the homopolymer distributions' standard deviations.

Therefore given an input FVDM, the ***flowgram*** module creates distinct FVDMs, derived from the input FVDM, to be used at different stages along a homopolymer progression.

For the sake of simplicity, the ***flowgram*** module only increases the standard deviations of Gaussian positive flow distributions. This is the reason for the restriction of positive flow distributions (in the user-specified flow value distribution model) to being Gaussian. It is simpler to increase the spread of a distribution if we restrict the type of the distribution we expect.

As an example of flow value degradation along a flow value progression, let us say the user specifies that the standard deviations of the Gaussian positive flow distributions should increase by 0.005 after every 10 flow cycles. Let us say that the user specifies that there should be 50 flow cycles in total for the simulation. Then we require 5 (50 divided by 10) distinct flow value distribution models. Then:

1. The FVDM used for the first 10 flow cycles will be the original user-supplied FVDM.

2. The FVDM used for flow cycles 11 to 20 will be derived by increasing the standard deviation of each Gaussian positive flow distribution (in the original FVDM) by 0.005

3. The FVDM used for flow cycles 21 to 30 will be derived by increasing the standard deviation of each Gaussian positive flow distribution (in the original FVDM) by 0.010 ($= 0.005 \times 2$)

4. The FVDM used for flow cycles 31 to 40 will be derived by increasing the standard deviation of each Gaussian positive flow distribution (in the original FVDM) by 0.015 ($= 0.005 \times 3$)

5. The FVDM used for flow cycles 41 to 50 will be derived by increasing the standard deviation of each Gaussian positive flow distribution (in the original FVDM) by 0.020 ($= 0.005 \times 4$)

From now on, we will refer to the list of FVDMs derived from the original user-supplied input FVDM as the compound FVDM.

As a summary:

- ***flowgram* Module Input**: A homopolymer progression, the number of MPP flow cycles to simulate, a user-defined FVDM and the two user-defined parameters that control flow value degradation:

    1. The Gaussian positive flow distribution standard deviation increase gradient (we store this in a variable ***sdIncFactor***)
    2. The number of flow cycles preceding each subsequent standard deviation increase (we store this in a variable ***sdInterimFlowCycles***)

- ***flowgram* Module Processing**: The processing carried out by the ***flowgram*** module is divided into two sub-steps:

1. Generating a sequence of distinct FVDMs (from the original user-supplied model) for use at different stages along the input homopolymer progression. This is the compound FVDM for MPP simulation.
2. Converting each homopolymer length in the input homopolymer progression into a flow value. This is done in two steps:
   (i) Identifying the number of the flow cycle to which the homopolymer in the progression belongs. Subsequently, given the flow cycle number, the specific FVDM (within the compound FVDM) that identifies with the flow cycle number is selected. This is the FVDM that describes the degree of flow value degradation at that flow cycle number.
   (ii) Given a selected FVDM, the homopolymer is converted into a flow value by deriving a random value from the distribution of flow values for that specific homopolymer length according to the FVDM.

- ***flowgram* Module Output**: There are two outputs for this module:
  1. The compound FVDM
  2. A flowgram derived from the input homopolymer progression

4.4.2 Algorithms and Data Structures

The ***flowgram*** module consists of one helper method, the ***incFunc*** method. The ***incFunc*** method takes a Gaussian distribution from the original FVDM and a factor $\boldsymbol{x}$ as input. Subsequently the method returns a Gaussian distribution whose standard deviation is greater than the input Gaussian distribution by (***sdIncFactor***) $\times$ $\boldsymbol{x}$.

The ***flowgram*** module uses the ***incFunc*** method to derive each FVDM in the compound distribution model. The module first calculates how many distinct FVDMs are necessary for simulation.

The number of FVDMs necessary for simulation depends on the number of flow cycles specified (let us assume this is stored in a variable ***cycles***) and the flow cycle degradation parameter ***sdInterimFlowCycles***.

Specifically, the number of distinct FVDMs in the compound distribution model is derived by rounding the value of $\frac{\boldsymbol{cycles}}{\boldsymbol{sdInterimFlowCycles}}$ up to the nearest integer. This is intuitive because a distinct FVDM is required to cater for each ***sdInterimFlowCycles*** number of flow cycles within ***cycles*** flow cycles.

The method call:

```
flowgram[homopolymerProg_,cycles_,distModel_,sdIncFactor_,
sdInterimFlowCycles_]
```

reads in a homopolymer progression (***homopolymerProg***), the number of flow cycles for MPP simulation (***cycles***), an FVDM (***distModel***) and the two flow cycle degradation parameters (***sdIncFactor*** and ***sdInterimFlowCycles***). We have established that the module returns a compound FVDM (derived from ***distModel***) as well as a flowgram derived from ***homopolymerProg*** using the compound FVDM.

The input FVDM ***distModel*** is represented by a 10×1 Mathematica List data structure containing 10 distribution specifications (for homopolymer lengths 0 to 9).

The outputted compound FVDM is represented by an $\boldsymbol{n}\times 10$ Mathematica List data structure containing $\boldsymbol{n}$ distinct FVDMs where $n = \frac{\boldsymbol{cycles}}{\boldsymbol{sdInterimFlowCycles}}$ and each FVDM stores 10 distribution specifications (for homopolymer lengths ranging from 0 to 9).

Similarly to a homopolymer progression, an outputted flowgram is represented as a $\boldsymbol{n}\times 2$ Mathematica List of flows, where $\boldsymbol{n}$ is the number of distinct flows in the flowgram and each flow is specified by:

1. A nucleotide base

2. The flow value corresponding to the nucleotide base

The Mathematica code for the ***flowgram*** module (including the ***incFunction*** method) is provided in appendix A.4 of this thesis.

Note that the complexity of the error function module is $\Theta(4 \times \boldsymbol{c})$, where $\boldsymbol{c}$ is the number of flow cycles for MPP simulation. This is because the module converts each homopolymer in the input homopolymer progression of length $4 \times \boldsymbol{c}$ into a flow value. The complexity of $\Theta(4 \times \boldsymbol{c})$ is equivalent to a complexity of $\Theta(\boldsymbol{c})$.

$\frac{NL}{l}$ fragments derived from the ***fragmentSequence*** module will result in $\frac{NL}{l}$ homopolymer progressions derived from the ***sequencingReaction*** module. Mapping the flowgram module onto each resulting homopolymer progression yields an overall complexity of $\Theta(\frac{NLc}{l})$ for simulation stage 3.

4.5 Simulation Stage 4: Flowgram Refinement

In the introductory section of this chapter we described the flowgram refinement stage of simulation as being composed of:

1. An optional sub-stage where ambiguous flow cycles are identified and trimmed from the input flowgram.

2. A sub-stage where the input flowgram is quality trimmed.

Let us discuss each sub-stage in detail.

4.5.1 Simulation Stage 4 Sub-Stage 1

4.5.1.1 Requirements and Specifications

We have established that this sub-stage consists of two primary objectives:

1. Identifying ambiguous flow cycles

2. Removing ambiguous flow cycles

We defined an ambiguous flow cycle as a flow cycle in which for all four nucleotides (T, A, C and G) flowed no significant light signal is observed. We therefore need a threshold flow value such that flow values below this threshold can be identified as not significant enough to register.

In theory, a good threshold would be the upper boundary of the negative flow distribution, because this distribution represents noise values (very low flow values that are not considered to originate from a chemical reaction) [7].

However, the negative flow value distribution in our default original FVDM is log-normal. Log-normal distributions are known to be infinitely-tailed.

We therefore have to devise a way to set an upper bound for a log-normal negative flow distribution. We can do this by only considering flow values that are above a particular frequency in the negative flow distribution. This is shown in Figure 4.2. This principle applies in a generic sense and can still be used for whatever distribution the user supplies for negative flow values. We still apply this principle, even if the user-supplied negative flow distribution is bounded, for the sake of uniformity.

Also for the sake of uniformity we use the same principle to bound the Gaussian positive flow distributions (This will be shown later).

For our simulation, we can therefore claim that an ambiguous flow cycle is a flow cycle in which all of the four distinct nucleotides flowed do not register a flow value greater than the upper boundary of the negative flow value distribution.

As a summary:

- ***Simulation stage 4 Sub-stage 1*** **Input**: A flowgram.
- ***Simulation stage 4 Sub-stage 1*** **Processing**: Identifying and trimming the ambiguous flow cycles and any flow cycles surrounded by ambiguous flow cycles in the input flowgram.

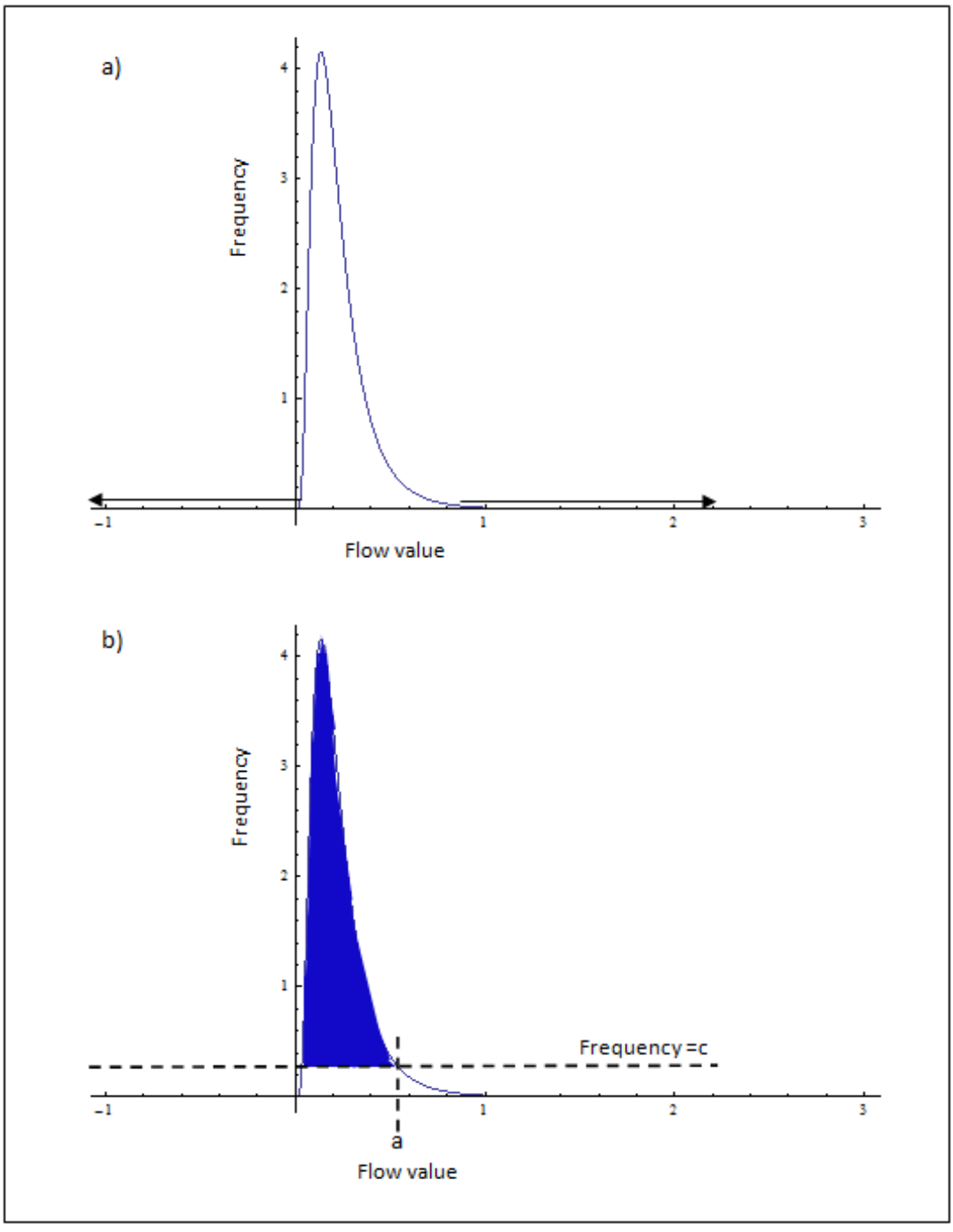

Figure 4.2: Constructing an upper bound for the (infinitely-tailed) default log-normal negative flow distribution - (a) The infinitely-tailed log-normal distribution. (b) If we only consider flow values that register a frequency above a certain threshold (the threshold is ***c*** in the diagram) then we only consider the shaded section of the log-normal distribution, whose upper boundary is the flow value ***a***

- ***Simulation stage 4 Sub-stage 1*** **Output**: A version of the input flowgram that contains no ambiguous flow cycles.

4.5.1.2 Algorithms and Data Structures

This sub-stage consists of 3 helper methods. The Mathematica code for these methods is provided in Appendix A.5.1.

1. **Method 1:** `overlapInterval[r_,s_]`

   This method is used in both the MPP simulation stage 4 sub-stages.

   The ***overlapInterval*** method takes two integers, r and s, as parameters. Subsequently it returns the interval of flow values for which the two distributions identified by the input parameters overlap.

   For the sake of simplicity, we only consider the overlap of the two distributions in the original (user-supplied) FVDM (and not the other FVDMs in the compound distribution model derived in simulation stage 3, because the interval of overlap varies in these as the distribution standard deviations are different).

   For the integers supplied as parameters to the ***overlapInterval*** method:

   (i) The integer 0 identifies the negative flow distribution in the original flow value distribution model.

   (ii) The flow value distribution for homopolymer length 1 is identified by 1

   (iii) The flow value distribution for homopolymer length 2 is identified by 2

   (iv) The flow value distribution for homopolymer length 3 is identified by 3

   And so on and so forth up to homopolymer length 9. In general, the flow value distribution for homopolymer length n is identified by n.

For example the invocation ***overlapInterval***[0,1] returns the interval of overlap between a negative flow distribution and the distribution for homopolymer length 1 in the original FVDM.

Let us however recall that the positive flow distributions are Gaussian. Gaussian distributions are infinitely-tailed distributions. Therefore an interval of overlap between any two Gaussian distributions is an infinite interval. To solve this problem, we have to bound our Gaussian positive flow distributions in the same way we bounded our negative flow distribution. For consistency, we apply the same threshold frequency in bounding all the distributions (including the log-normal distribution) in our FVDM. This fixed threshold frequency can take on any arbitrary value (We selected a value of 0.05. Therefore given a homopolymer length distribution from our FVDM, we only consider flow values that register a frequency over 0.05).

The ***overlapInterval*** method is of complexity $\Theta(1)$.

Note: As we shall see in section 4.8, PyroSim invokes the ***overlapInterval*** method with parameters 0 and 1. As a result the public variables ***inv1***, ***inv2*** and ***inv*** store information as follows:

(a) ***inv1*** stores the interval of flow values in the negative flow distribution that have a frequency above 0.05

(b) ***inv2*** stores the interval of flow values in the distribution for homopolymer length 1 that have a frequency above 0.05

(c) ***inv*** stores the interval of overlap between the negative flow distribution and the distribution for homopolymer length 1.

These public variables are used by the ***showAmbiguousCycles*** and ***qualityTrim*** modules.

2. **Method 2:** `showAmbiguousCycles[flowgram_]`

The ***showAmbiguousCycles*** method takes a flowgram, ***flowgram***, as an input parameter. The method identifies and replaces any ambiguous flow cycles in ***flowgram*** with the symbol {N}.

The ***showAmbiguousCycles*** method is of complexity $\Theta(\boldsymbol{c})$, where $\boldsymbol{c}$ is the number of flow cycles for MPP simulation.

3. **Method 3:** `trimAmbiguousCycles[flowgram_]`

The ***trimAmbiguousCycles*** method takes a flowgram, ***flowgram***, as an input parameter. Note that each ambiguous flow cycle in the input flowgram ***flowgram*** is denoted by the symbol {N} at this point. The ***trimAmbiguousCycles*** method removes all the ambiguous flow cycles in ***flowgram*** as well as any flow cycles surrounded by ambiguous flow cycles.

The ***trimAmbiguousCycles*** method is of order $\Theta(1)$.

A mapping of the ***showAmbiguousCycles*** and ***trimAmbiguousCycles*** methods onto each flowgram in a list of $\frac{NL}{l}$ flowgrams results in an overall complexity of $\Theta(\frac{NLc}{l})$ for simulation stage 4 sub-stage 1.

4.5.2 Simulation Stage 4 Sub-Stage 2

4.5.2.1 Requirements and Specifications

We have discussed quality trimming in detail in subsection 2.1.5.1 of this thesis. PyroSim implements a ***qualityTrim*** module. In addition to the flowgram to be trimmed, the module takes two parameters that dictate the quality trimming process:

1. The maximum proportion of flow values in a flow value progression that can be in the overlap region (between the negative flow value distribution and the distribution of flow values of homopolymer length 1) for the progression to be

considered a high quality progression. Let us say that this is stored in a variable ***errThreshold***

2. The minimum length for a quality-trimmed flowgram. Let us say that this is stored in ***lengthThreshold***

In order to trim a pool of flowgrams, PyroSim maps the ***qualityTrim*** module, which trims an individual flowgram, onto each flowgram in the pool.

As a summary:

- ***qualityTrim* Module Input**: A flowgram and quality trimming parameters ***errThreshold*** and ***lengthThreshold***

- ***qualityTrim* Module Processing**: The input flowgram is trimmed until either the proportion of flow values in the overlap region in question is less than ***errThreshold*** or the length of the flow value progression is less than ***lengthThreshold***. If the case is the latter then the resulting flowgram is filtered out of the pool of high quality flowgrams.

- ***qualityTrim* Module Output**: A quality-trimmed flowgram or **Null** if the flowgram is found to be of poor quality

4.5.2.2 Algorithms and Data Structures

The Mathematica code for the ***qualityTrim*** module is provided in Appendix A.5.2.

The complexity of the quality trim module is $\Theta(L^2_{flowgram})$, where $L_{flowgram}$ is the length of the input flowgram.

If the ambiguous cycles in the input flowgram have not been identified and trimmed, $L_{flowgram} = 4 \times \boldsymbol{c}$ (because the length of the flowgram is still equal to the length of the underlying homopolymer progression derived by the ***sequencingReaction*** module).

If however the ambiguous cycles in the input flowgram have been trimmed, $L_{flowgram}$ is some fraction of its original value, i.e. $L_{flowgram} = C_1 \times 4 \times \boldsymbol{c}$ for some fractional constant C_1.

Therefore we can always express $L_{flowgram}$ as being proportional to the number of flow cycles set for simulation. By extension, we can express the complexity of the ***qualityTrim*** module as being approximately $\Theta(\boldsymbol{c}^2)$

A mapping of the ***qualityTrim*** module onto each flowgram in a list of $\frac{NL}{l}$ flowgrams results in an overall complexity of $\Theta(\frac{\boldsymbol{NLc}^2}{l})$ for simulation stage 4 sub-stage 2.

4.6 Simulation Stage 5: Producing flowgram graphs for the flowgram

4.6.1 Requirements and Specifications

We have established that one of the sequencing outputs from an MPP instrument is a series of graphs, each graph pertaining to an observed progression of flow values.

Our simulator should incorporate a ***viewFlowgrams*** module, which draws and displays a flowgram graph for each quality trimmed flowgram.

4.6.2 Simulation Stage 5 Algorithms and Data Structures

The ***viewFlowgrams*** module takes a list of all the trimmed flowgram as a parameter (***flowgramList***). The module implements a ***drawFlowgram*** method which draws a single flowgram graph (derived from a single input flowgram). The ***viewFlowgrams*** module maps the ***drawFlowgram*** method onto each flowgram in ***flowgramList***.

The ***drawFlowgram*** method takes an individual flowgram as a parameter (let us say that this is stored in a variable ***flowgramData***) and uses the colors red, yellow, blue and green to represent the nucleotides T, A, G and C respectively, for each flow

cycle. The method draws a bar for each flow value, where the height of a bar is directly proportional to the flow value, such that the relative heights of the bars in the flowgram graph represent the flow values in the flowgram.

The Mathematica code for the viewFlowgrams module is provided in Appendix A.6.

We estimate the complexity of the ***viewFlowgrams*** module to be $\Theta(N_f \times L_{flowgram})$ where N_f is the total number of flowgrams for which the module is called and $L_{flowgram}$ is the average flowgram length. This is equivalent to a complexity of $\Theta(N_f \times c)$. Further, the number of flowgrams, N_f is proportional to the number of flowgrams prior to the flowgram refinement stage, which is $\frac{NL}{l}$). We can therefore restate the complexity of the ***viewFlowgrams*** module as $\Theta(\frac{NLc}{l})$

Some examples of flowgrams graphs generated for flowgrams by PyroSim are shown in Figure 4.3.

4.7 Simulation Stage 6: Base Calling and Quality Scores

4.7.1 Requirements and Specifications

Base calling is an integral part of the MPP methodology. It involves the translation of each flow value (in a quality-trimmed flowgram) into the most likely homopolymer length that produced the flow value.

Determining the underlying homopolymer length that produces a flow value is not intuitive, due to the ambiguous nature of flow values in MPP.

Base calling translates a flow value into an underlying homopolymer length through the utilization of the Bayesian Equation 2.1 listed in subsection 2.1.5.2. We previously established that the utilization of this equation requires a FVDM.

Recall, however, that a compound FVDM was used in simulation stage 3 to translate homopolymer lengths into flow values. It follows that accurate base calling would also require the use the compound FVDM. This is such that given a flow value (within a

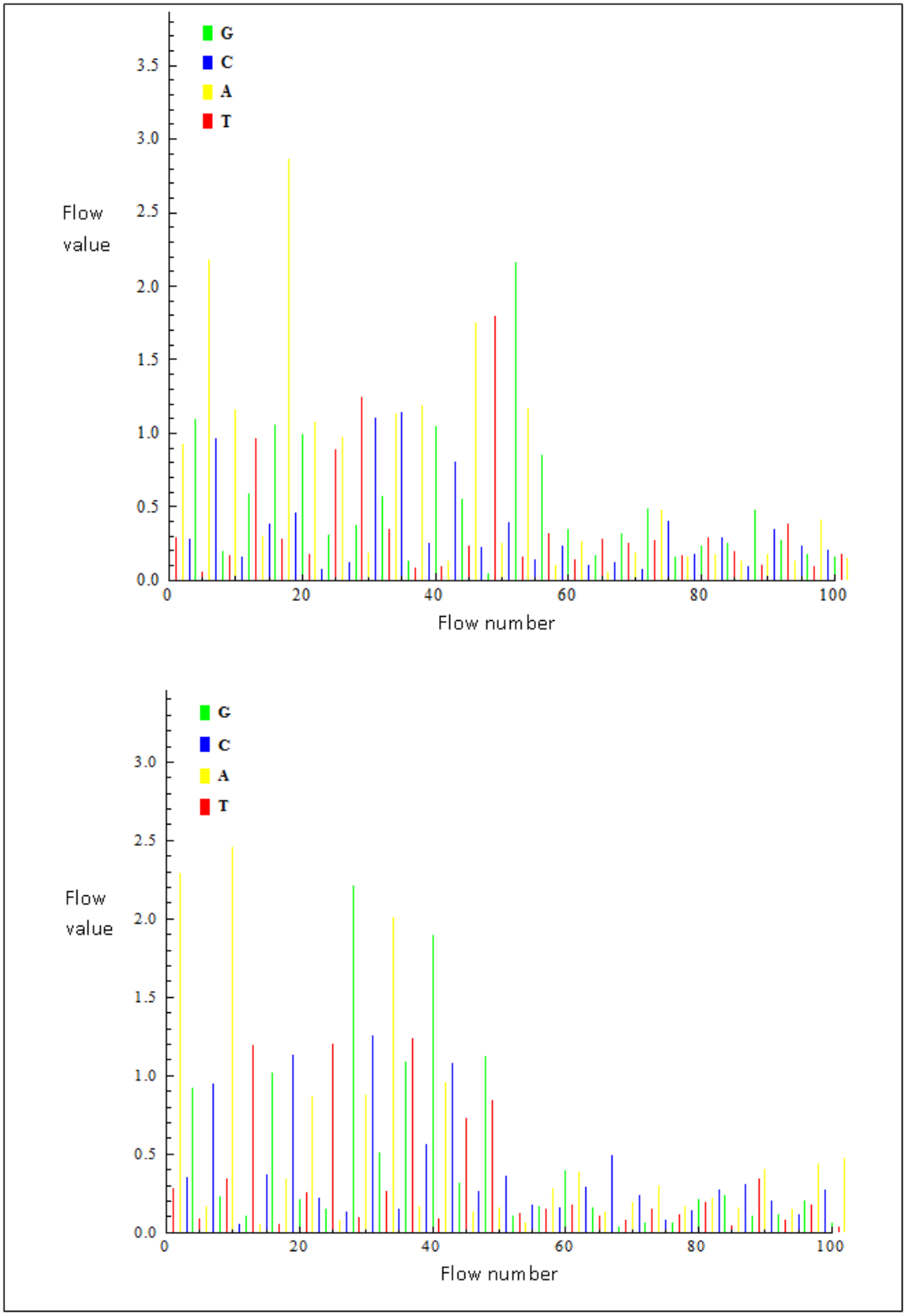

Figure 4.3: Example flowgrams outputted by the viewFlowgrams module of our simulator - The key for DNA base representation in a flowgram always accompanies the flowgram in PyroSim output

flowgram) determining the most likely underlying homopolymer length is carried out in the steps listed below:

(i) Determine the number of the flow cycle (within the flowgram) that the flow value belongs to. Let us call this ***flowCycleNum***.

(ii) Determining the FVDM (within the compound model) that indicates the flow value degradation at ***flowCycleNum***. Let us call this model ***seletedM***

(iii) Determining the most likely homopolymer length that produced the flow value (by using Equation 2.1), according to ***selectedM***.

The underlying homopolymer progression estimated by base calling is subsequently translated into a sequencing read. We discussed base calling in detail in subsections 2.1.5.2 and 3.1.1.2 of this thesis.

Having derived a sequencing read, we subsequently assign quality scores to each DNA nucleotide in the read. The quality score of a nucleotide is a measure of the confidence that the homopolymer length at the position of the nucleotide in the read is correct. Quality scoring is discussed in detail in subsection 2.1.5.3 of this thesis.

We implement base calling of an individual flowgram and the assignment of quality scores to the derived nucleotide sequence in the ***baseCall*** module of our simulator.

PyroSim performs base calling on a pool of flowgrams by mapping the ***baseCall*** module onto each flowgram in the pool.

As a summary:

- ***baseCall* Module Input**: A quality-trimmed flowgram, the compound FVDM and the flow cycle degradation parameter ***sdInterimFlowCycles***

- ***baseCall* Module Processing**: The input flowgram is translated into a homopolymer progression by base calling. Subsequently the resulting homopolymer

progression is translated into a sequencing read. Quality scores are assigned to each base in the sequencing read.

- ***baseCall* Module Output**: A sequencing read along with a list of quality scores for each base in the sequencing read.

4.7.2 Simulation Stage 6 Algorithms and Data Structures

We devise algorithms for the ***baseCall*** module by viewing stage 6 of simulation as having two primary objectives:

Objective 1: Base calling - which we have established to be the interpretation of a flowgram into an underlying homopolymer progression

Objective 2: Quality scoring

Objective 1 is implemented by the use of 3 helper methods, which collectively implement Equation 2.1. The Mathematica code for these methods is provided in Appendix A.7. The three helper methods are:

1. `pHomopolymerLengthN[n_]`

   This method takes a homopolymer length, $\boldsymbol{n}$, as a parameter and returns the prior probability of $\boldsymbol{n}$.

2. `pSgivenN[s_,n_,flowValueDistModel_]`

   This method reads in a flow value, $\boldsymbol{s}$, a homopolymer length, $\boldsymbol{n}$ and an FVDM, ***flowValueDistModel***, as input parameters. The method returns the likelihood of $\boldsymbol{s}$ given $\boldsymbol{n}$ according to the FVDM ***flowValueDistModel***.

3. `pNgivenS[n_,s_,flowValueDistModel_]`

   This method reads in a flow value, $\boldsymbol{s}$, a homopolymer length, $\boldsymbol{n}$ and an FVDM, ***flowValueDistModel***, as input parameters. The method returns the posterior

probability of ***n*** given ***s*** according to ***flowValueDistModel*** by implementing Equation 2.1.

The ***pHomopolymerLengthN***, ***pSgivenN*** and ***pNgivenS*** methods are public methods in the PyroSim package, allowing a PyroSim user to call each of these methods directly. These methods are all of complexity $\Theta(1)$.

The ***baseCall*** module performs base calling by determining the most likely underlying homopolymer length (i.e. the homopolymer length that yields the highest posterior probability given a flow value) for each flow value in an input flowgram, ***flowgram***.

Recall that the ***flowgram*** module uses a compound FVDM to obtain flowgrams. It follows that accurate base calling will require the use of a compound FVDM, so that the specific FVDM that indicates the flow value degradation at each flow value along ***flowgram*** is used to translate the flow value into an underlying homopolymer length.

The ***baseCall*** module therefore reads in two additional input parameters:

1. ***compoundDistModel*** - The compound FVDM as an input parameter

2. ***sdInterimFlowCycles*** - An indicator of how many interim flow cycles precede each increase in flow value degradation

We now discuss objective 2 of the ***baseCall*** module.

Recall, from subsection 2.1.5.3, that quality scoring utilizes Equations 2.2 and 2.3.

Because PyroSim utilizes FVDMs that range up to homopolymers of length 9, we can restate Equation 2.3 for our purposes as Equation 4.1. The ***baseCall*** module employs Equations 4.1 and 2.2 to quality score each nucleotide in a sequencing read derived from base calling.

The Mathematica code for the ***baseCall*** module is provided in Appendix A.7. The complexity of the ***baseCall*** module is $\Theta(L_{flowgram}) \approx \Theta(c)$, where $L_{flowgram}$ is the length of the input flowgram.

A mapping of the baseCall module onto each flowgram in a list of approximately $\frac{NL}{l}$ flowgrams results in an overall complexity of $\Theta(\frac{NLc}{l})$ for simulation stage 6.

Equation 4.1:

$$Pr = \sum_{j=n}^{9} P(j|s)$$

- ***Pr*** - Probability of the base at position ***n*** in the homopolymer being correct. This is equivalent to the probability of a homopolymer of at least length ***n*** given a flow value ***s***. ***Pr*** is obtained by summing of the posterior probabilities of homopolymers of with length ***n*** and greater, given ***s***.

4.8 PyroSim: Use of constituent modules to achieve MPP simulation

PyroSim reads in a number of parameters as input. The parameters can be set to their default values by calling PyroSim's initialization methods.

Below is a list of PyroSim's input parameters:

1. The ***population*** parameter (Variable name: ***population***) - This is the input population/list of nucleotide sequences for sequencing simulation.

2. The ***FVDM*** (Variable name: ***flowValueDistModel***) - This parameter serves as the template for the compound FVDM used for MPP simulation. It is a sequential list of distributions for homopolymer lengths 0 to 9. For our simulator, positive flow distributions (distributions for homopolymer lengths ≥ 1) are constrained by the fact that they must be Gaussian.

The default ***FVDM*** is version 2 of the Richter et. al. FVDM.

3. The ***flow cycle*** parameter (Variable name: ***flowcycles***) - This is the number of flow cycles to simulate.

4. The ***distribution*** parameters - These parameters are utilized for stage 1 of MPP simulation. Specifically the parameters characterize the distribution of the lengths of the DNA fragments outputted from this simulation stage. These ***distribution*** parameters are:

   (i) A distribution type (Variable name: ***distribution***). Our simulator restricts this to two options, either `Normal` or `Uniform`.

   (ii) Parameter 1 (Variable name: ***param1***). This is the first parameter for the distribution identified by ***distriburion***. If ***distribution*** is set to `Normal Distribution` then ***param1*** is interpreted to store the mean of ***distribution***. Otherwise if ***distribution*** is set to `Uniform Distribution` then ***param1*** is interpreted to store the lower bound of ***distribution***.

   (iii) Parameter 2 (Variable name ***param2***. We abbreviate this as ***param2***). This is the second parameter for the distribution identified by ***distribution***. If ***distribution*** is `Normal Distribution` then ***param2*** is interpreted to store the standard deviation of ***distribution***. Otherwise if ***distribution*** is `Uniform Distribution` then ***param2*** is interpreted to store the upper bound of ***distribution***. Note that if ***distribution*** is `Uniform Distribution` then ***param1*** must be less than or equal to ***param2***.

5. The ***flow value degradation*** parameters - These parameters are used to generate the compound FVDM used for MPP simulation and subsequently for base calling. The ***flow value degradation*** parameters:

   (i) The Gaussian positive flow distribution standard deviation increase gradient

(Variable name: ***deg***).

(ii) The interim number of flow cycles that precede each Gaussian positive flow distribution standard deviation increase (Variable name: ***degFactor***).

For example if ***deg*** = 0.005 and ***degFactor*** = 10 then, in the compound FVDM, the standard deviation of the Gaussian distribution for homopolymer length ***n*** will increase by 0.005 every 10 flow cycles (For all $1 \leq n \leq 9$).

6. The ***trim ambiguous flow cycles*** parameter (Variable name: ***trimA***) - This is a boolean that may be set to either `True` or `False` indicating whether or not to identify and trim ambiguous flow cycles respectively.

7. The ***quality trimming*** parameters - These parameters dictate the quality trimming of flowgrams, which occurs in simulation stage 4 sub-stage 2. The quality trimming parameters are:

   (i) The maximum proportion of flow values (in a flowgram) that should be in the interval of overlap between a negative flow distribution (homopolymer length 0) and a unit base incorporation distribution (homopolymer length 1) for the progression to be considered a high quality progression. This parameter is stored in a variable named ***qtErr***.

   (ii) The threshold length that a flowgram should not fall below during quality trimming. This parameter is stored in a variable named ***qtThreshold***.

8. The ***output specification*** parameter (Variable name: ***out***) - This parameter may be set to `Flow Value Progressions` or `Sequencing Reads`. If ***out*** is set to `Flow Value Progressions`, then PyroSim outputs quality trimmed flow value progressions as the final output of MPP simulation. Alternatively if ***out*** is set to `Sequencing Reads` then PyroSim outputs sequencing reads.

9. The ***print flowgrams*** parameter (Variable name: ***printFlowgram***) - This is a boolean that may be set to either `True` or `False` indicating whether or not to print MPP flowgrams graphs respectively. A flowgram graph is a graph of the sequence of flow values in a quality trimmed flowgram.

10. The ***logFile*** parameter. This is the full name of the file to which the log of the PyroSim MPP simulation will be written.

11. The ***flowgramFile*** parameter. If ***printFlowgram*** is set to `True`, this is the full name of the file to which the output MPP simulation flowgrams will be exported.

An invocation of PyroSim takes the following format:

```
result =
PyroSim[population, flowValueDistModel, flowcycles, {distribution, param1,
    param2}, {deg, degFactor}, trimA, {qtErr, qtThreshold}, out,
        printFlowgram, logFile, flowgramFile]
```

The Mathematica code that dictates how PyroSim uses its constituent modules to achieve MPP simulation is provided in Appendix A.8.

Note:

1. PyroSim generates output in the form of MPP flowgrams or MPP sequencing reads, depending on the user's specification. This is evident in Figure 4.1.

   Each outputted flowgram or sequencing read is accompanied by a positional vector indicating its position of origin. In addition to this, the specific variant in the input population that the flowgram or read originates from is also outputted.

2. We have established that the total complexity of performing each distinct stage of simulation, with the exceptional of quality trimming, is approximately $\Theta(\frac{NLc}{l})$. Recall that the complexity of quality trimming is $\Theta(\frac{NLc^2}{l})$.

We conclude that the overall complexity of PyroSim is equal to the complexity of the most computationally expensive simulation stage. The complexity of a PyroSim simulation is therefore $\Theta(\frac{NLc^2}{l})$.

4.9 A comparison of PyroSim and Flowsim

Recall that our MPP simulator PyroSim is based on the MPP simulation model utilized in the Flowsim simulator.

An important aspect of PyroSim is flexibility of simulation parameters. PyroSim reads in the FVDM for MPP simulation from the user, unlike Flowsim, where the FVDM is fixed. We opted for this approach because of the contention over the true distribution of flow values about homopolymer lengths in MPP FVDMs (Refer to Subsection 3.1.2). A PyroSim user can also specify the parameters for flow value degradation.

The authors of Flowsim do not explicitly mention a provision for trimming ambiguous flow cycles. Ambiguous flow cycles within an MPP read compound the problem of determining the true origin of the read (Refer to Subsection 6.1.2).

Flowsim produces simulation output in the form of SFF files, which are the files outputted by 454 sequencing instruments in reality. In future we hope to consolidate simulation output from PyroSim into SFF files.

The authors of Flowsim do not provide a complexity analysis of the simulator.

5

MPP Read Alignment - Implementation and Verification

We implemented the following MPP read alignment algorithms:

1. Sequence-space alignment algorithms:
    - Smith-Waterman alignment
    - Asymmetric Smith-Waterman alignment
    - Gap Propagation
    - PyroAlign
2. Flow-space alignment algorithms:
    - Flowgram Matching

Recall that these algorithms are described in subsection 3.2.1 of this thesis.

All the algorithms are implemented in a Wolfram Mathematica 8.0 package called MPPAlign. MPPAlign also contains public auxiliary functions that facilitate MPP read alignment. The full code for MPPAlign is available in the Mathematica package.

MPPAlign can be accessed in two ways:

1. Via the MPPSoft web graphical interface at the URL:

   `http://hammer.cs.ukzn.ac.za/~anisa/`

2. Via the Mathematica 8.0 package, MPPAlign, which can be downloaded at the URL:

   `http://hammer.cs.ukzn.ac.za/~anisa/MPPAlign.html`

The complete MPPAlign user manual is available at the URL:

`http://hammer.cs.ukzn.ac.za/~anisa/MPPAlignManual.pdf`

This chapter explains how we verified the correctness of our algorithm implementations and further provides a complexity analysis of the algorithms implemented in MPPAlign.

5.1 Algorithm Verification

For the algorithm verification and experiments conducted in the remainder of this thesis (i.e. for chapters 5-8):

1. We use a length of 150 bp as the standard length of our reference sequence. We consider the length 150 bp to be a fair trade-off considering that some of the algorithms we have implemented are computationally expensive and the time taken to run them increases with the lengths of the reference sequence and the MPP reads.

2. The reference sequences are subsequences (of length 150 bp) of HIV pol sequences obtained from Genbank. Genbank is a comprehensive public nucleotide sequence database [24].

The specific reference sequences that we used for our experiments are available in Appendix D of this thesis.

Recall that in subsection 2.2 of this thesis we described sequencing the HIV pol quasispecies as an example of the use of the MPP methodology to sequence a heterogeneous population of nucleotide sequences. We therefore conduct our analysis of the MPP read alignment and MPP error correction algorithms on sequence data that encapsulates the type of data that these algorithms would be used to process in reality.

Note that typically in MPP experiments to determine the variation in the HIV pol gene, a 1 kb region of the gene is sequenced [3]. Our experiments are therefore a parallel of MPP experiments on a small scale.

Note further that we do not specifically identify the mutations in the HIV pol nucleic acid sequence that are responsible for drug resistance. Instead we look at identifying authentic variation in 150 bp subsequences of HIV pol in a generic sense.

3. We have observed that the sequencing read coverage of the genomic region corresponding to the reference sequence is a function of three parameters:

   (i) The number of sequencing reads

   (ii) The lengths of the sequencing reads

   (iii) The spread of the sequencing reads along the genomic region

   In order to obtain coverages sufficient enough for us to be able to make deductions from our experiments with a reasonable number of sequencing reads, we have chosen to set the parameters for PyroSim such that the mean fragment length of the fragments derived in PyroSim stage 1 is relatively long - approximately $\frac{1}{6}$ of 150 bp (the length of the reference sequence). This is a mean fragment length

of 25 bp. It follows that simulation will result in sequencing reads with a mean length of roughly 25 bp.

Given this, the MPP reads that correspond to a reference sequence are derived in two steps:

(i) Generating a population of one or more variants of the reference sequence by using PyroSim's ***GeneratePopulation*** module.

(ii) Running a simulation on PyroSim with the population obtained from (i) as input and the following parameter settings:

- ***flowcycles*** = 12 (We discovered that this sets a sufficient number of flow cycles for a fragment length of 25 bp. This number of flow cycles ensures that the simulated flows do not exceed short fragment lengths by a big margin leading to long chains of ambiguous flow cycles at the tail end of derived flowgrams. Also, given this number of flow cycles, the simulated flows are not too few leading to the loss of information from long fragment lengths)
- ***distribution*** = `Normal`
- ***param1*** = $\frac{1}{6} * 150 = 25$ (This sets the mean fragment length)
- ***param2*** = 5.5 (A decent variation in fragment lengths means that there is more variation in the positions that the fragments originate from. This is closer to the reality of MPP)

The rest of PyroSim's input parameters are set to their default values.

Note: For each experiment we generate a total of 300 reads. This is achieved by deriving a total of 50 sequences from the ***GeneratePopulation*** module (whereby each variant produces approximately 6 reads as the mean fragment length is $\frac{1}{6}$ of 150 bp, the standard variant sequence length), which are subsequently input into PyroSim.

We now discuss how we verified the correctness of our implementations of the MPP read alignment algorithms.

5.1.1 Smith-Waterman alignment

We implemented the original SW alignment algorithm described in [20].

We compare our implementation of SW alignment with the Bioconductor implementation of SW alignment in order to verify the correctness of the former. Bioconductor is widely utilized open source software suite for computational biology and bioinformatics [23].

Our comparison has two objectives:

i) To ensure that each SW local alignment of a sequencing read with a reference sequence produced by our algorithm yields an identical or equivalent alignment to the one produced by the Bioconductor SW algorithm.

ii) To ensure identical alignment scores between our SW algorithm and the Bioconductor SW algorithm.

Let us define an aligned sequencing read as a sequencing read that may or may not contain gaps as a result of having been aligned with the reference genome.

Let us also define a SW alignment trial as the alignment of a sequencing read with a reference sequence by the Bioconductor SW algorithm to obtain an aligned sequencing read ***a1*** and the alignment of the same sequencing read with the same reference sequence by the MPP align SW algorithm to obtain an aligned sequencing read ***a2***.

We achieve objective 1 by implementing a scoring scheme in the comparison of ***a1*** and ***a2***. In the scoring scheme, given a position, ***pos***, along ***a1***, if the nucleotide (or gap) at position ***pos*** in ***a1*** is not identical to the nucleotide (or gap) at position ***pos*** in ***a2***, ***pos*** is assigned a score of -1, otherwise (if the nucleotide at ***pos*** in ***a1*** is equal to the

nucleotide at ***pos*** in ***a2***) ***pos*** is assigned a score of 0. The scores for all positions along the length of ***a1*** are tallied to produce an overall score. If ***a1*** and ***a2*** are identical, the overall score should be 0.

We conducted 300 distinct SW alignment trials. In the trials we used 15 distinct reference sequences, conducting 20 trials for each reference sequence. We derived the sequencing reads corresponding to a reference sequence by inputting a homogeneous population of 20 copies of a variant of the reference sequence into PyroSim. For each reference sequence the reads were derived for variants of mutation distance 0.03, 0.05, 0.10 and 0.17. Five trials were conducted for each pair of reference sequence and mutation distance.

Our trial results are shown in Figure 5.1. We combined the results for all 300 trials into Figure 5.1 because the results that we observed for the distinct reference sequence and mutation distance pairs were similar.

In Figure 5.1 (a), we observe that the alignments produced by the MPP align SW algorithm are not always identical to the Bioconductor SW algorithm alignments. The two algorithms however produce equivalent alignments, evidenced by the identical alignment scores in Figure 5.1 (b). Given a trial, we observed that any discrepancies between ***a1*** and ***a2*** were due to the MPP align SW algorithm initiating the alignment from the end points of the reference sequence and sequencing read, while the Bioconductor algorithm possibly initiates the alignment from the starting points of the reference sequence and sequencing read.

Given our SW alignment trial results, we conclude that our SW algorithm is a correct implementation of SW alignment. Note that we conducted our SW alignment trials with the parameter set outlined in [20], this being: match = 1, mismatch = $-\frac{1}{3}$, gap open penalty = -1, gap extend penalty = $-\frac{1}{3}$.

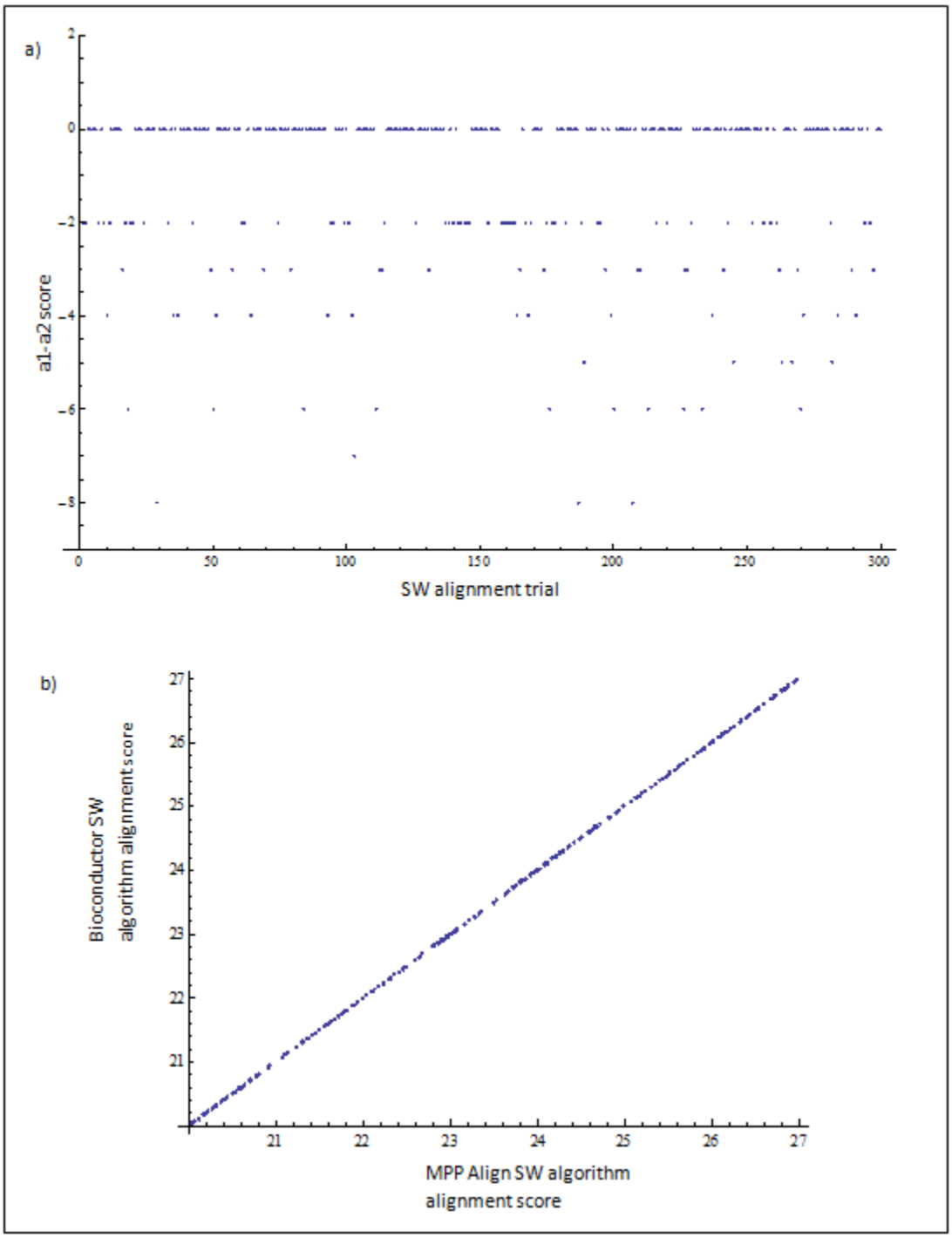

Figure 5.1: SW alignment trial results - (a)The a1-a2 scores for all 300 trials are shown. We note that in most trials a1 and a2 are identical, yielding an a1-a2 score of 0. (b)The alignment scores for all 300 trials are shown. We observe that for every trial, the MPP align SW alignment score is identical to the Bioconductor SW alignment score.

5.1.2 Asymmetric Smith-Waterman alignment

We have established that ASW alignment is a modification of SW alignment.

We implemented the ASW algorithm described in [13] by modifying our SW algorithm in two steps:

1. Modifying the alignment parameters to the following: match = 1, mismatch = -3, gap open penalty = -5, gap extend penalty = -2.

2. Applying a weight, $\boldsymbol{W}$ (derived by Equation 3.4), asymmetrically to the sequencing read (and not the reference sequence).

We surmise that our ASW algorithm is a correct implementation of the algorithm described in [13] because it is a modification to our implementation of SW, which we have verified to be correct.

5.1.3 Gap Propagation

Our Gap Propagation algorithm is executed in two steps:

1. Each sequencing read in the list of reads produced by an MPP experiment is pairwise-aligned to the reference sequence by a fixed pairwise alignment strategy. The pairwise alignment strategy is determined by the user. The user has the option between the Smith Waterman and Semi-Global pairwise alignment schemes. The same pairwise alignment strategy is applied consistently to each sequencing read in the list of reads.

   From now on we will abbreviate the term Semi-Global pairwise alignment by SG.

2. Any gaps that may have been inserted into the reference sequence during the execution of step 1 are propagated through the aligned reads that overlap the reference sequence at that alignment position.

Recall that Gap Propagation produces a multiple alignment of sequencing reads, such that the reads are aligned relative to each other in addition to each read being aligned relative to the reference sequence.

The quality of the final multiple alignment that results from gap propagation depends on the quality of the intermediate step of the algorithm (the individual pairwise alignments of each sequencing read with the reference genome). We therefore allow a user to specify the parameters for the intermediate pairwise alignment strategy utilized in step 1 of the algorithm, in addition to specifying the pairwise alignment strategy itself.

In the description of Gap Propagation in [4], the authors do not provide links to or information about a software implementation of the algorithm with which we can compare our implementation.

Our approach to verifying our algorithm is based on the theory of Gap Propagation.

We surmise that step 1 of our algorithm is executed correctly because we have already verified the correctness of SW alignment algorithms.

In order to verify our implementation of SG, the second option for step 1 of Gap Propagation, we performed similar experiments to the SW alignment trials in subsection 5.1.1. A comparison of our SG alignment algorithm with the Bioconductor implementation of SG established that our alignment algorithm produces alignment results equivalent to the latter.

We now need to verify that the propagation of gaps through the sequencing reads that constitutes step 2 of the algorithm is executed in a correct fashion. In order to do this we conduct an experiment as described below.

Let us define an aligned reference sequence as a version of the reference sequence that may or may not contain gaps as a result of the pairwise SW or SG alignment of the latter with a sequencing read.

We previously defined an aligned sequencing read as a sequencing read that may or may not contain gaps as a result of pairwise SW (or ASW or SG) alignment with the reference sequence.

Therefore when we pairwise align a sequencing read to the reference sequence, gaps are inserted at appropriate positions in both sequences to yield a maximal alignment score, and consequently the sequencing read becomes an aligned sequencing read and the reference sequence becomes an aligned reference sequence.

Therefore the pairwise alignment of each read, r_i, in a list of $\boldsymbol{n}$ sequencing reads in step 1 of Gap Propagation results in an aligned sequencing read a_i and a corresponding aligned reference sequence ref_i.

The aim of step 2 of Gap Propagation is to propagate the gaps in an aligned reference sequence, ref_i, to all the aligned sequencing reads, $a_{1...n}$, that overlap ref_i at the gap positions. This is done for all reference sequences in $ref_{1...n}$.

For the sake of our experiment, during propagation, each time we insert a gap into an aligned sequencing read a_i, we insert a gap into the corresponding reference sequence ref_i.

Given this, perfect Gap Propagation would result in all the reference sequences in $ref_{1...n}$ being exactly identical at the end of stage 2. This is because if Gap Propagation is executed perfectly, any gap that occurs in an aligned reference sequence ref_i in step 1 is propagated correctly through all the other reference sequences in $ref_{1...n}$ (in addition to all the aligned sequencing reads in $a_{1...n}$).

Therefore in order to prove that our Gap Propagation algorithm is correct, we are required to prove that when the algorithm has finished executing, all the reference sequences in $ref_{1...n}$ are identical.

We define a Gap Propagation trial as an experiment where we align a list of sequencing reads to a corresponding reference sequence by utilizing our Gap Propagation algorithm. Given the list $ref_{1...n}$ resulting from a trial, we devise a scoring scheme

to establish if all the sequences in $ref_{1...n}$ are equal. In the scoring scheme we iterate through the columns in $ref_{1...n}$. Given a column, we obtain a non-redundant list of all the distinct nucleotide bases in the column. If the sequences in $ref_{1...n}$ are identical, this list should contain strictly one nucleotide (or gap) for all the columns in $ref_{1...n}$. Our scoring scheme begins with a score (which we call the ID score) of 0 and deducts -1 for each extra nucleotide base in the list of distinct nucleotide bases for each column. If an ID score of 0 is retained after iterating through the columns of $ref_{1...n}$, we conclude that the sequences in $ref_{1...n}$ are identical.

We conducted 300 Gap Propagation trials, which were constituted by 20 trials for each reference sequence in a list of 15 distinct reference sequences. As with our SW alignment trials described in subsection 5.1.1, we derived the sequencing reads corresponding to a reference sequence by inputting a homogeneous population of 20 copies of a variant of the reference sequence into PyroSim. Given a reference sequence, reads were derived variants of mutation distance 0.03, 0.05, 0.10 and 0.17. Five trials were conducted for each pair of reference sequence and mutation distance.

Our Gap Propagation trial results are shown in Figure 5.2. Given that all the ID scores from our experiment are 0, we surmise that the sequences in $ref_{1...n}$ are always equal after gap propagation and by extension, we surmise that our algorithm is a correct implementation of Gap Propagation.

5.1.4 PyroAlign

Our implementation of PyroAlign is based on the algorithm described by Saeed et. al. in [4]. There is a software implementation of PyroAlign available on Saeed's webpage: http://multimedia.ece.uic.edu/fahad/professional.html.

In conducting several multiple sequencing read alignment runs (whereby each run was conducted by aligning the reads from a PyroSim MPP simulation with a corresponding reference genome), we discovered that the PyroAlign algorithm implemented

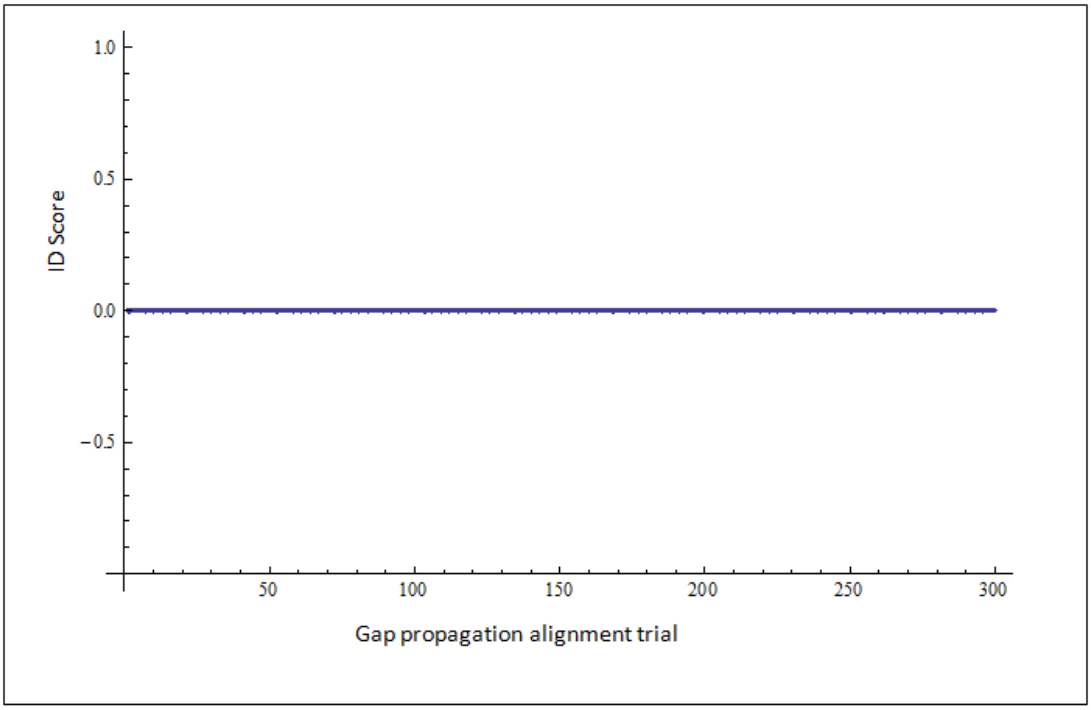

Figure 5.2: Gap Propagation alignment trial results - The ID scores for all 300 trials are shown in the graph above. We note that the scores are all 0

by the authors does not produce consistently good quality multiple alignment solutions. Specifically, the quality of the solution is highly influenced by the length of the reference genome as well as the spread and number of the sequencing reads.

Since the software implementation of PyroAlign failed to produce satisfactory results, we found it useful to implement a version of PyroAlign that allows a user to tweak the alignment parameters during the different stages of alignment.

We noted that there are some inconsistencies between the author's implementation of PyroAlign and the algorithm described in [4]. For instance the domain decomposition strategy outlined in [4] is not the strategy used to construct a hierarchical multiple alignment of sequencing reads in the software implementation. Further in [4], the authors state that after the reordering stage, each pair of neighboring sequencing reads is pairwise aligned using the standard Needleman Wunsch algorithm. However the software implementation of PyroAlign pairwise aligns each pair of neighboring sequencing reads using the Smith Waterman algorithm. Our perception of PyroAlign is that there does not appear to be a stable algorithm for the former. Further, the author's implementation of the algorithm does not produce the multiple sequencing read alignment quality shown in the paper.

Our implementation of PyroAlign adheres to the author's software implementation of the algorithm.

We verify our implementation of PyroAlign by verifying each of the different stages of the algorithm. Here we perceive PyroAlign as being comprised of four distinct stages:

1. Stage 1 - Semi-global alignment of each sequencing read with the reference genome.

   In our implementation of PyroAlign, a user may alternatively opt to use the SW local alignment algorithms for the pairwise alignment of each sequencing read with the reference genome. We have already verified our implementations of SW and SG in subsections 5.1.1 and 5.1.3 respectively.

2. Stage 2 - Reordering the aligned sequencing reads according to their starting position (with respect to the reference sequence) after the semi-global alignments. This stage is straight forward.

3. Stage 3 - SW pairwise alignment of each pair of neighboring sequencing reads. This step is also verified by virtue of our SW verification in subsection 5.1.1.

4. Stage 4 - Hierarchial pairwise profile-profile alignments. The author's implementation of PyroAlign calls the external program clustalW2 to execute pairwise profile-profile alignments. clustalw2 is an open source sequence alignment program [26]. Our implementation of PyroAlign also calls the external program clustalw2 to execute pairwise profile-profile alignments.

5.1.5 Flowgram Matching

Given an individual flowgram from an MPP experiment, our flowgram matching algorithm aligns the flowgram with respect to a flow space representation of the reference genome, using enhanced suffix arrays as an optimization as described in [18].

Recall that Flowgram Matching is a flow-space algorithm whereas the SW and ASW alignment, Gap Propagation and PyroAlign are sequence-space algorithms.

Since Flowgram Matching aligns individual flowgrams with respect to the reference genome, we consider it a pairwise alignment algorithm rather than a multiple alignment algorithm.

Flowgram Matching returns a series of alignment positions of an input flowgram with respect to an input flow-space representation of the reference genome.

The returned alignment positions all indicate possible positions of origin of the input flowgram with respect to the reference genome. The outputted alignment positions are accompanied by their corresponding alignment scores. Returned alignment positions with higher corresponding scores are more likely to pinpoint the true origin of the

flowgram with respect to the reference genome.

Following [18], each alignment position returned by our algorithm is returned because it meets the criteria of yielding a score (calculated according to Equation 3.12) that is above a specific threshold score.

We provide a user with the option of specifying the threshold score directly or using a p-value to derive a threshold score. A p-value can be used to set the threshold score according to the algorithm descried in [19]. The p-value determines the accuracy of the alignment positions returned by the Flowgram Matching algorithm for an input flowgram. A smaller p-value corresponds to a higher threshold score and therefore more accurate albeit fewer returned alignment positions.

Our implementation of Flowgram Matching also reads in an FVDM used to evaluate the likelihood of a flow value given a homopolymer length. The calculation of this likelihood is part of Equation 3.11.

The instructions on how to supply parameters to our flowgram matching algorithm are available in the MPPAlign manual.

To verify that we optimized Flowgram Matching correctly, we conducted an experiment to compare the alignment positions (and corresponding alignment scores) produced by our algorithm with the alignment positions and scores produced by a brute force approach, where Equation 3.12 (in chapter 3 of this thesis) was computed for all $\boldsymbol{m}$-long segments in the flow-space representation of the reference genome.

We conducted 200 Flowgram Matching trials, whereby a flowgram matching trial is defined by using our algorithm to derive alignment positions for an individual flowgram with respect to the reference sequence and doing the same via the brute force approach. We varied the reference genome sequence and conducted fresh runs of PyroSim to derive MPP flowgrams for each run. We also varied the variants of the reference genome sequence (used to derive MPP flowgrams) between 0.03, 0.05, 0.10 and 0.17.

Our results showed that the alignment position scoring information delivered by

our implementation of Flowgram Matching is correct. The alignment position that was found to yield the maximal score for an input flowgram by our Flowgram Matching algorithm, during a trial run, was also identical to the alignment position that yielded the maximal score in the brute force approach (for the same flowgram).

5.2 Complexity Analysis

Our implementation of SW involves the pairwise SW alignment of an individual sequencing read of length L_R to a reference genome sequence of length L_Γ. Given a total of $\boldsymbol{N}$ reads, this yields an overall time complexity of $\Theta(NL_RL_\Gamma)$. This is the same time complexity for SW alignment described in literature [4].

Our ASW algorithm is essentially version of SW with different parametrization. Therefore ASW has a complexity of $\Theta(NL_RL_\Gamma)$.

Gap Propagation involves the pairwise alignment of each sequencing read (in a list of $\boldsymbol{N}$ reads) with the reference genome. This is stage 1 of the algorithm, with a complexity of $\Theta(NL_RL_\Gamma)$. Subsequently the propagation of gaps through the aligned sequencing reads has a complexity $\Theta(N^2)$. This yields an overall complexity of $\Theta(NL_RL_\Gamma + N^2)$ This is conversant with the time complexity of the algorithm stated in [4].

The total time complexity of stage 1 of PyroAlign is $\Theta(NL_RL_\Gamma)$, where N is the number of sequencing reads input into the algorithm, L_Γ is the length of the reference genome sequence and L_R is the mean sequencing read length.

From Appendix B.4.3, we estimate the total time complexity of stage 2 of PyroAlign to be $\Theta(N)$.

In stage 3 of PyroAlign, we conduct pairwise alignment for each pair of neighboring reads. At the beginning of stage 3, each read will have a length directly proportional to the length of the reference sequence, following pairwise alignment to the reference sequence in stage 1. The pairwise alignment of each pair of neighboring reads, where

each read is of length L_Γ is of time complexity $\Theta(L_\Gamma^2)$. Given a total of N reads, there will be $\frac{N}{2}$ read pairs in total. This gives stage 3 an overall complexity of $\Theta(\frac{N}{2}L_\Gamma^2)$, which is equivalent to a complexity of $\Theta(NL_\Gamma^2)$.

We were unable to find an estimate of the time complexity for clustalw2 pairwise profile alignments utilized in stage 4 of PyroAlign. In [4], the authors state that the total complexity of stage 4 of PyroAlign should be roughly $\Theta(NLogN \times L_\Gamma^2)$.

We therefore estimate a total complexity of our implementation of PyroAlign to be $\Theta(NL_RL_\Gamma + N + NL_\Gamma^2 + L_\Gamma^2 NLogN)$. This is asymptotically equal to $\Theta(NL_RL_\Gamma + NL_\Gamma^2 + L_\Gamma^2 NLogN)$.

Flowgram Matching has an overall complexity that is sub-linear to $\Theta(NL_RL_\Gamma)$. This is because the algorithm involves the pairwise alignment of each flowgram (of length proportional to L_R) in a list of N MPP flowgrams to a flow-space representation of the reference genome (of length proportional to L_Γ). However, due to the implementation of enhanced suffix arrays, sub-linear complexity is achieved.

The variations in execution times for our alignment algorithms given different parameterizations are shown in Figures 5.3, 5.4 and 5.5.

From the above figures, it is evident that Flowgram Matching consistently presents the best execution times. Further, Flowgram Matching is seen to be the most scalable algorithm. On the other hand, PyroAlign consistently presents the worst execution times and is the least scalable algorithm. We however note that the author's of PyroAlign designed the algorithm for parallelization [4]. The results we report are due to the execution of each listed algorithm on a single core (no parallelization). The experiments to determine the execution times for our experiments were performed on a 2.60GHz Dual-Core CPU with 2.00 GB of RAM.

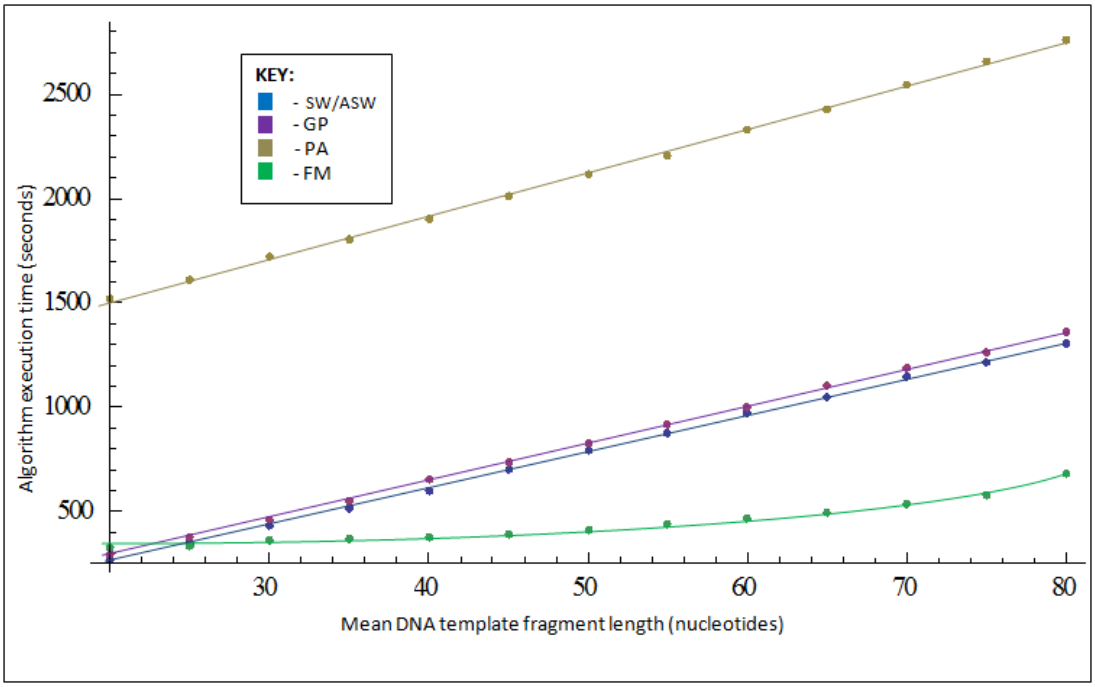

Figure 5.3: Dependence of read alignment algorithm execution time on mean DNA template fragment length ($\approx \mathbf{L}_R$) - We see that as predicted by the time complexity, the relationship between the execution time for Smith Waterman Alignment (SW) and the mean DNA template fragment length is linear. The same applies to Asymmetric Smith Waterman Alignment (ASW), Gap Propagation (GP) and PyroAlign (PA). GP and PA have longer execution times proportional to the additional steps in the algorithms. The execution time for Flowgram Matching (FM) has a sub-linear relationship with the mean DNA template fragment length. These experiments were performed with a fixed number of reads ($L_R = 200$) and a fixed reference genome nucleotide sequence length (L_Γ=150)

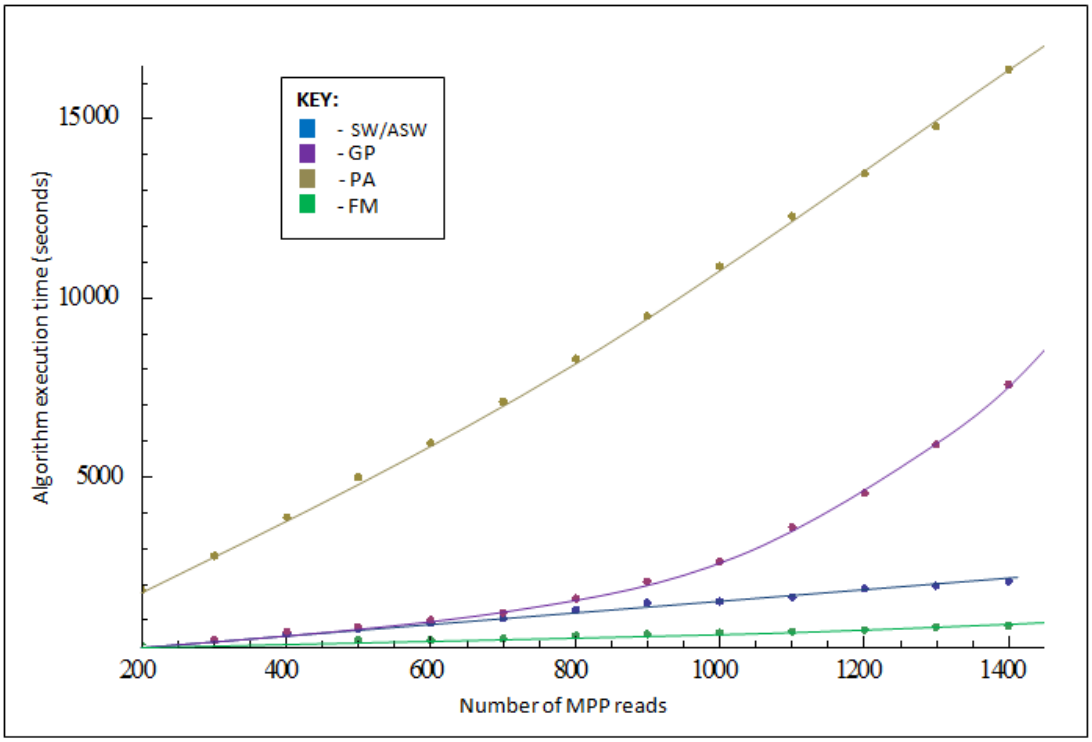

Figure 5.4: Dependence of read alignment algorithm execution time on number of input MPP reads - As predicted by the time complexity, the relationship between the execution time for SW/ASW and the number of reads is linear. The relationship between the execution time for GP and the number of reads is seen to be quadratic. This agrees with the complexity of GP. PA and FM have $NlogN$ and sub-linear relationships with the input number of reads respectively. These experiments were performed with a fixed reference genome nucleotide sequence length ($L_\Gamma = 150$) and a fixed mean DNA template fragment length ($L_R \approx 25$).

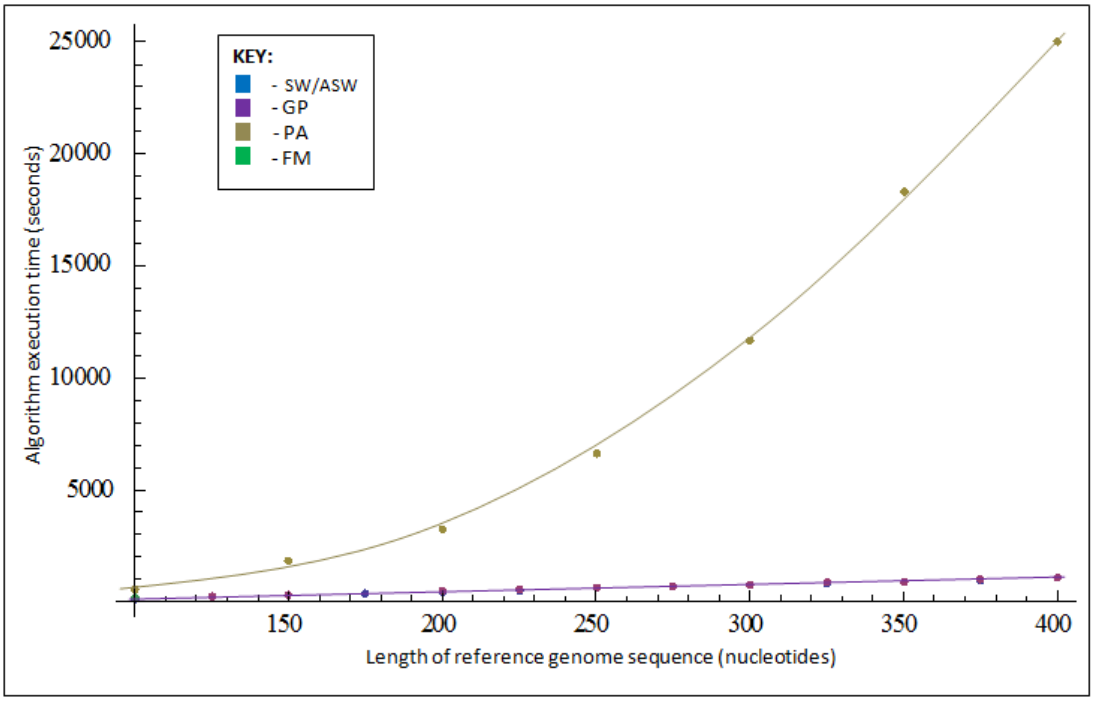

Figure 5.5: Dependence of read alignment execution time on length of reference genome nucleotide sequence - The relationship between the execution time for SW/ASW and the length of the reference genome is linear. Further the relationship between the execution time for FM and the reference genome length is sub-linear. The curves for SW/ASW and FM are not visible here. We see that the GP curve is linear. Further the PA curve is quadratic. The described observations are conversant with the algorithm complexities. These experiments were performed with a fixed number of reads ($L_R = 200$) and a fixed mean DNA template fragment length ($L_R \approx 25$).

6

MPP Read Alignment - Qualitative Comparison

Given a list of reads from an MPP experiment, we have established there to be two levels of MPP read alignment, these being:

1. The pairwise alignment of each MPP read with respect to the reference genome.
2. The subsequent alignment of a list of MPP reads (whereby each read in the list has been individually aligned with respect to the reference sequence during level 1 of MPP read alignment) with respect to each other.

SW and ASW alignment as well as Flowgram Matching only complete the first level of MPP read alignment.

Both Gap Propagation and PyroAlign have an intermediate stage that completes the first level of MPP read alignment, followed by the subsequent completion of the second level of MPP read alignment.

This chapter is organized as follows:

In section 6.1 we begin our analysis by determining the best strategy for pairwise alignment of each MPP read with the reference genome. We compare SW and ASW

alignment, SG alignment (which is an option for the level 1 pairwise alignment strategy in our implementation of the PyroAlign and Gap Propagation algorithms) and Flowgram Matching.

Following this, in section 6.2, we conduct a comparative qualitative analysis of the overall alignment solutions delivered by SW and ASW alignment, Flowgram Matching, PyroAlign and Gap Propagation.

6.1 MPP Read Pairwise Alignment Strategies - Qualitative Comparison

We conduct an experiment to compare the SW, ASW and SG alignment algorithms as well as Flowgram Matching.

We provide the results of the experiment and make a conclusion based on these results.

Note that we use the standard alignment parameters for SW given in [20] and for ASW given in [13]. We use the same alignment parameters for SG as SW in order to enable a fair comparison between the two algorithms.

6.1.1 Experiment Design

Recall that MPP read is a generic term used to describe both a sequencing read and an MPP flowgram. Recall also that sequencing reads constitute the raw data in a sequence-space analysis whereas MPP flowgrams constitute the raw data in a flow-space analysis.

Let us outline the fundamental difference between sequence-space alignments and flow-space alignments.

We have established that in sequence-space, sequencing errors directly lead to a variation in the lengths of the homopolymers in a sequence read with respect to the

lengths of the corresponding homopolymers in the original fragment from which the sequencing read is derived.

Specifically, sequencing errors lead to augmented or diminished homopolymers in the sequencing read. In the pairwise alignment of a sequencing read with the reference genome sequence, gaps should be inserted into the sequencing read to fill in the missing nucleotides in a diminished homopolymer in the sequencing read. Similarly, gaps should be inserted into the reference genome sequence such that the gaps are aligned with the extra nucleotides in an augmented homopolymer in the sequencing read. Examples of this are shown in Figure 6.1.

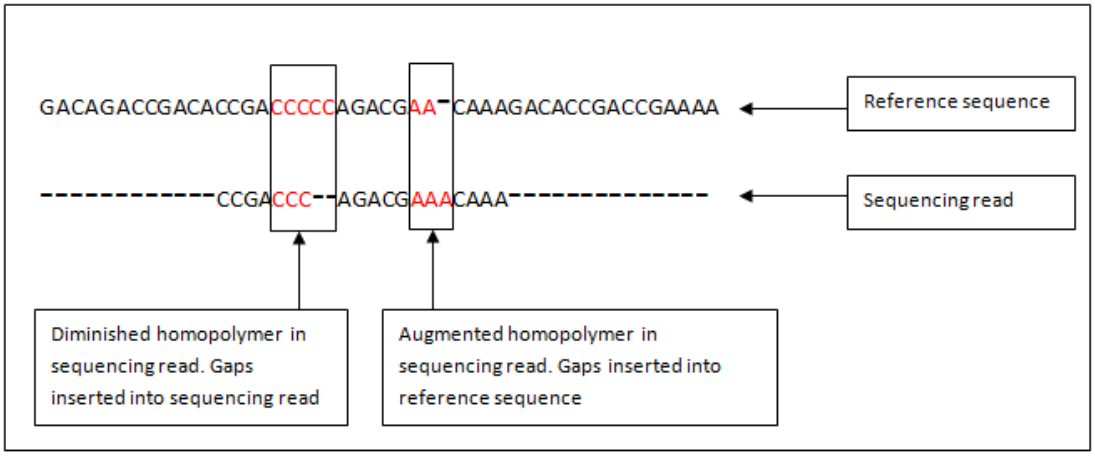

Figure 6.1: An example pairwise alignment of a sequencing read containing augmented and diminished homopolymers with a reference genome sequence - The number of gaps inserted into the sequencing read should correspond to the degree of diminution of the corresponding homopolymer. Similarly, the number of gaps inserted into the reference sequence should correspond to the degree of augmentation of the corresponding homopolymer

On the other hand, in flow-space augmented and diminished flow values are constrained by flows within flow cycles and therefore do not directly affect the length of a flowgram. An MPP flowgram outputted by PyroSim may however be shorter than the underlying homopolymer progression from which it was derived due to the flowgram refinement stage of sequencing simulation (which involves trimming ambiguous flow

cycles and quality trimming).

Further in flow-space, a flowgram derived from a fragment that does not begin with a T and end with a G is padded with non-authentic negative flows appended to both its front end and its tail end because the flowgram is specified in terms of complete flow cycles. (However note that the last cycle in the flowgram may be incomplete because some flows are shaved off during flowgram refinement).

Therefore, in general the starting and end points of an MPP read (i.e. a sequencing read or MPP flowgram) with respect to the reference sequence may be displaced slightly forward or slightly backward as compared to the original starting and end points of the read with respect to the reference sequence (both in sequence-space and flow-space).

We claim that a pairwise alignment algorithm is accurate if it can correctly estimate the true origin of an MPP read with respect to the reference genome. In sequence-space this corresponds to estimating the true origin of a sequencing read with respect to the reference genome nucleotide sequence whereas in flow-space this corresponds to estimating the true origin of a flowgram with respect to the flow-space representation of the reference genome.

A vast majority of the augmented or diminished homopolymers in a sequencing read (as shown by the example in Figure 6.1) will be due to sequencing error and not authentic variation. Recall that this is because in using PyroSim's ***generatePopulation*** module, authentic variants of the reference genome (from which the MPP reads are derived) are obtained through substitutions of nucleotide bases, and not by insertions and deletions of nucleotide bases (Refer to section 4.1). The unlikely case of an authentic variant of the reference sequence with an augmented homopolymer would occur if the nucleotides immediately following or preceding the homopolymer are substituted with a base identical to the base constituting the homopolymer in the variant. Similarly the unlikely case of an authentic variant with a diminished homopolymer would occur if the nucleotides at the head or tail of the homopolymer are substituted with a different

base to the base constituting the homopolymer in the variant.

Given the above argument, in sequence-space we claim that a pairwise alignment algorithm is accurate if through the alignment of a sequencing read with the reference sequence, augmented or diminished homopolymers in the sequencing read, which will largely be due to sequencing error, are obviated as shown in Figure 6.1. When an augmented homopolymer follows directly after a diminished homopolymer, then we have a sequencing substitution error. Such errors are more difficult to obviate, because a substituted nucleotide in a sequencing read with respect to the reference sequence could equally be due to authentic variation.

In the qualitative comparison of MPP read pairwise alignment strategies, we conduct an experiment that has two objectives:

Objective 1: Determining how well a given MPP read pairwise alignment algorithm estimates the true origin of the MPP read with respect to the reference genome.

Objective 2: Specifically for sequence-space pairwise alignment algorithms - Determining how well a sequence-space pairwise alignment algorithm obviates augmented and diminished homopolymers.

We carry out objective 1 by taking advantage of a feature of our simulator PyroSim. Recall that in chapter 4, we established that PyroSim accompanies each outputted MPP read with a {start, end} positional vector indicating the origin of the read with respect to the reference genome.

In order to carry out both objectives of our experiment, we use PyroSim to produce a list of MPP reads from a single variant of the reference genome.

We then use an MPP read alignment algorithm to position each of the reads outputted by PyroSim with respect to the reference genome.

To achieve objective 1, we compare the positioning of each MPP read with respect

to the reference genome by an MPP read alignment algorithm with the true origin position of the read. Our accuracy metric for an MPP read alignment algorithm is the percentage of MPP reads that are aligned within or very close to their true original positions with respect to the reference genome by the algorithm.

To achieve objective 2, we determine the overall percentage of augmented or diminished homopolymers in the aligned reads from an MPP experiment that are obviated by the sequence-space read alignment algorithm. In theory, if all the sequencing reads are perfectly aligned, the percentage of obviated augmented/diminished homopolymers in the set of sequencing reads should be 100%.

We evaluate objectives 1 and 2 for variants of mutation distances 0.03, 0.05, 0.10 and 0.17 from a reference genome sequence.

For objective 1 we examine SW, ASW and SG alignment as well as Flowgram Matching.

Similarly for objective 2 we evaluate the same algorithms excluding the flow-space alignment algorithm, Flowgram Matching.

Given an MPP read alignment algorithm and a variant distance from the reference genome, we perform 100 distinct alignments in order to conclusively evaluate objectives 1 and 2 for the specific algorithm and the specific variant distance from the reference genome. Each alignment is conducted with a fresh variant (of the same distance from the reference genome) and a fresh run of PyroSim to obtain MPP reads. We also incorporate 15 distinct reference sequences in our experiment.

6.1.2 Results and Discussion

The results from our experiment are shown in Figures 6.2 and 6.3 below.

Figure 6.2 presents the results for objective 1 of our experiment. We observe here that all the sequence-space alignment algorithms (SW, ASW and SG) perform well.

Further we observe that SG consistently maps the largest proportion MPP reads to

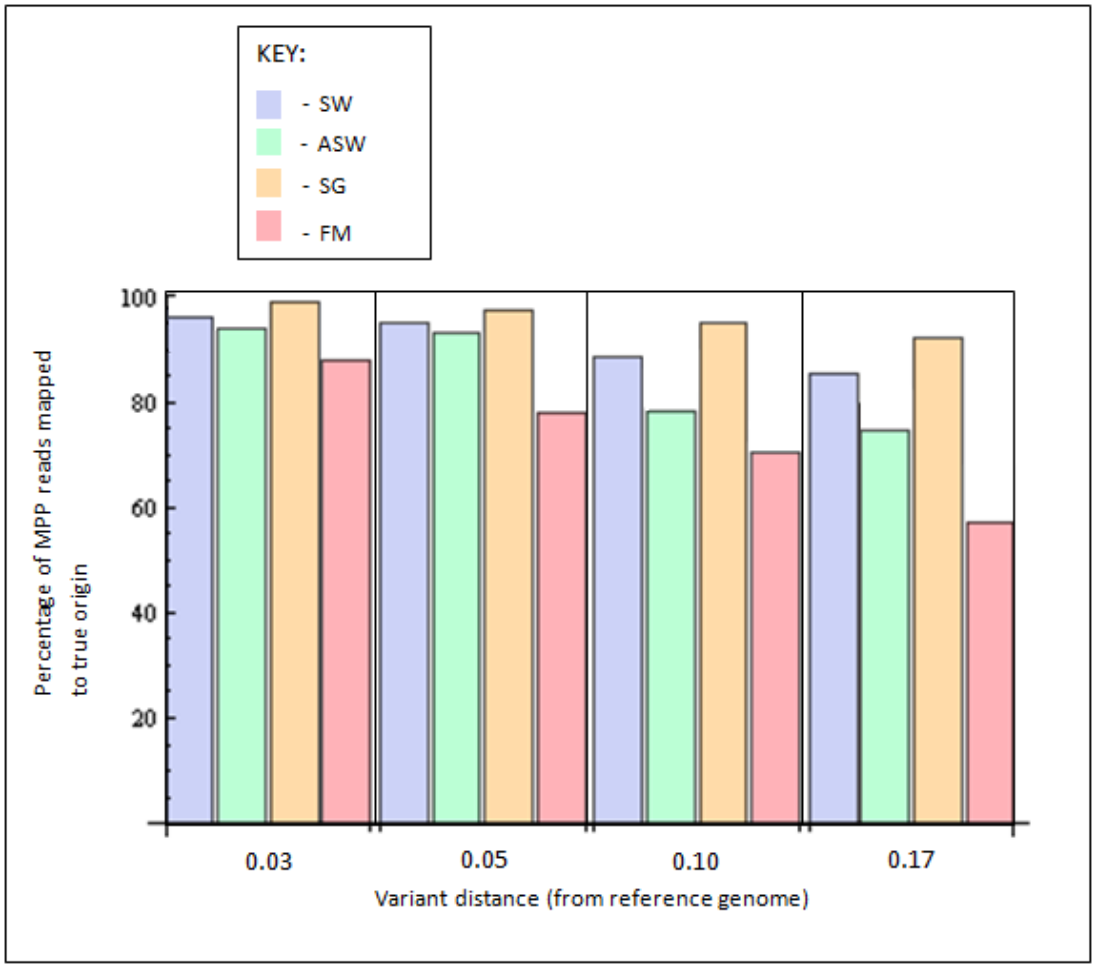

Figure 6.2: Experiment results for objective 1 - The results show the percentage of MPP reads mapped onto their true origin given an MPP read alignment algorithm and a variant mutation distance from the reference sequence

their true original positions (with respect to the reference genome) for variant mutation distances of 0.03, 0.05, 0.10 and 0.17. SW alignment is the second best performing MPP read alignment algorithm in this regard. During the conduction of our experiment, we observed that the SG and SW alignments are often identical and yield identical scores, aside from a small proportion of instances, where the SG alignments outperformed the SW alignments for the same set of reads. We attribute this to the established fact that SG alignment ignores the leading and the trailing gaps when aligning a sequencing read to a reference sequence. This ensures that the algorithm is more sensitive to the alignment of the sequencing read itself to a subsequence of the reference sequence.

The relatively poor performance of Flowgram Matching (abbreviated as FM) as compared to the sequence-space algorithms is contrary to the argument presented in literature ([18],[15],[17],[7]) which postulates that working in flow-space produces better solutions than working in sequence-space.

In observing Flowgram Matching pairwise alignments we noted two important points:

1. Flowgram Matching is highly sensitive to the number of flow cycles used to derive the MPP reads (which are input into the Flowgram Matching algorithm) from our simulator PyroSim.

   The more the number of flow cycles used to derive the input MPP reads/flowgrams exceeds the DNA nucleotide sequence fragment lengths (in stage 2 of PyroSim MPP simulation), the less accurate the Flowgram Matching pairwise alignments.

   A flowgram derived from an MPP simulation with excess flow cycles will have additional non-authentic flow values (derived from the excess flows), which do not correspond to homopolymers in the reference genome, appended onto its tail end.

   In the alignment of such a flowgram with respect to a flow-space representation

of the reference genome, the score of placing the flowgram in its true origin is diminished because the alignment of the entire flowgram involves the alignment of non-authentic flows to homopolymers in the reference genome.

In such a case the flowgram may end up being aligned to a position in the reference genome that is not its true origin because the false alignment yields a better score.

Therefore in order to obtain more accurate results from flowgram matching, one should ensure that on average the number of flow cycles employed is just sufficient to cover the DNA nucleotide sequence fragment lengths. However flowgrams derived from very short fragments will still contain non-authentic flows.

Contrary to this, given a sequence-space alignment algorithm (SW, ASW, Gap Propagation or PyroAlign), the pairwise alignment accuracy of the algorithm is not as highly compromised by the number of flow cycles used to derive the input MPP reads.

Recall that the sequence-space alignment algorithms use sequencing reads as raw data as opposed to flowgram matching which uses flowgrams as raw data. Recall also that PyroSim derives sequencing reads from flowgrams by base calling. Base calling removes the effect of excess flow cycles in a sequencing read because non-authentic flow values are mostly translated into homopolymer lengths of 0, meaning that the sequencing read is not likely to have several non-authentic nucleotide bases appended onto its tail end. This means that the alignment of the sequencing read into its true original position with respect to the reference sequence is not as highly compromised.

2. Flowgram Matching is highly sensitive to variation with respect to the reference sequence.

   Recall that our experiment involves deriving MPP reads from variants of the reference sequence. This means that given an MPP read, the homopolymers from

which the flow values in the read are derived differ slightly from the homopolymers in the reference sequence. This diminishes the score of aligning the read into its true original position with respect to the reference sequence.

The consequence of this is the read may end up being aligned into a position that is not its true origin because the false alignment yields a better score.

On the other hand the sequence-space alignment algorithms do not exhibit the same sensitivity (given the alignment parameters that we utilized for the experiment). Authentic variation in a sequencing read will mostly result in the read being aligned to its true origin with respect to the reference sequence. This is such that any variant nucleotides are simply considered mismatches (substitutions) with respect to the reference sequence.

Figure 6.3 presents the results for objective 2 of our experiment, we observe that semi-global alignment consistently identifies the most augmented/diminished homopolymers for mutation distances 0.03, 0.05, 0.10 and 0.17. This is followed by SW alignment and subsequently ASW alignment. This indicates that semi-global alignment is the best algorithm for pairwise sequence-space alignments.

Our further observations, evident in Figure 6.3, are that ASW does not work better than SW, as described in [13]. In fact our experiments show that it is consistently the poorest performing sequence-space alignment algorithm.

In both Figure 6.2 and 6.3, the accuracy of each of the alignment algorithms decreases with the increase in the genetic distance of the variant from the reference sequence. This is an expected result as an MPP read produced by a more distant variant has a greater total variation from the reference genome, which leads to a less accurate alignment of the read with respect to the reference genome.

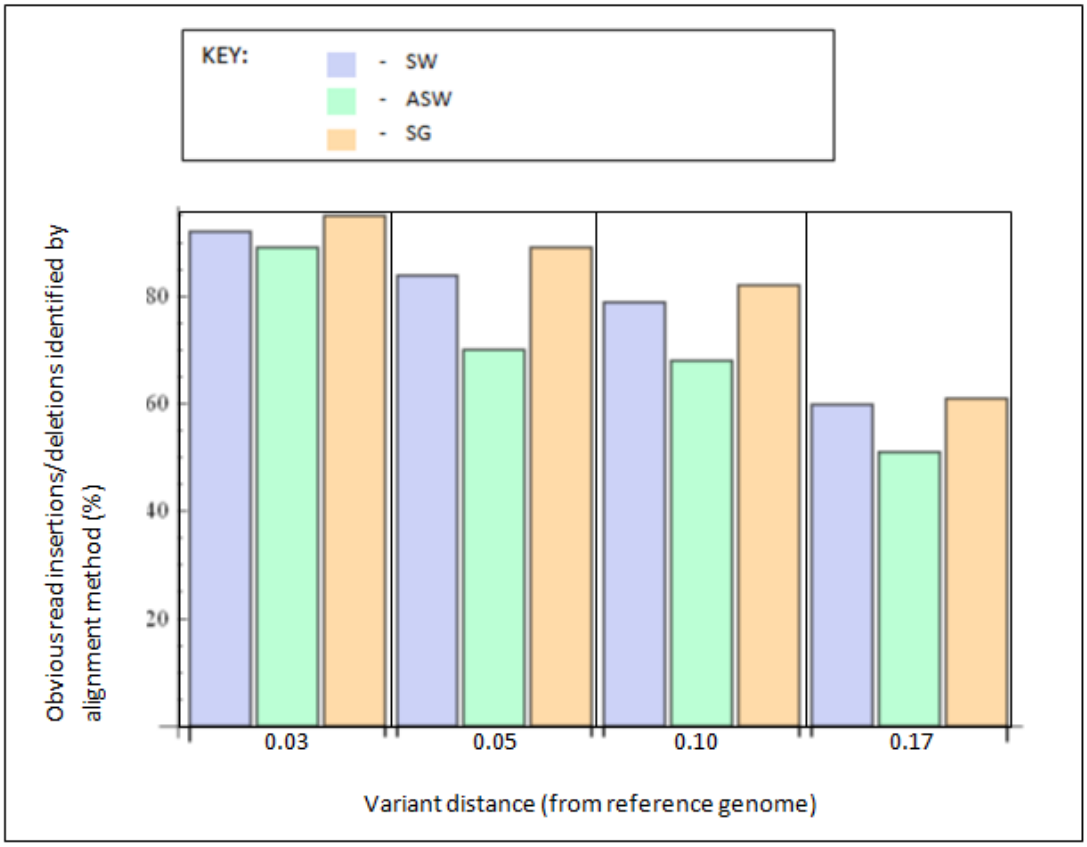

Figure 6.3: Experiment results for objective 2 - The results show the trend of the percentage of obvious augmented/diminished homopolymer scores identified for three sequence-space pairwise alignment algorithms (SW and ASW alignment as well as semi-global alignment abbreviated here as SG)

6.1.3 Conclusion

Our experiments establish that in the conduction of pairwise alignments of MPP reads with respect to the reference genome, flow-space algorithms do not necessarily yield more accurate results than sequence-space algorithms.

Particularly, because the quality of the pairwise alignment solution delivered by Flowgram Matching is very sensitive to the nature of the inputs to the algorithm, Flowgram Matching becomes a less reliable algorithm than its sequence-space alignment counterparts.

Further, we establish semi-global alignment to be a superior algorithm for sequence-space pairwise MPP read alignments.

6.2 Overall MPP Read Alignment - Qualitative Comparison

Similarly to the quality assessment of PyroAlign in [4], our overall quality assessment for each of the distinct MPP read alignment algorithms consists of two objectives:

1. **Overall Quality Assessment Objective 1**: An assessment of the algorithm's ability to place an MPP read in the correct context with respect to the reference genome.

2. **Overall Quality Assessment Objective 2**: An assessment of the algorithm's ability to handle the correct placement of MPP reads produced from sequencing multiple variants of the same genomic sequence.

This section conducts an overall alignment quality assessment for SW and ASW alignment, Gap Propagation, PyroAlign and Flowgram Matching.

Recall that in the previous section, we established semi-global alignment to be the best sequence-space pairwise alignment algorithm. We therefore use semi-global align-

ment for the intermediate stage that conducts the pairwise alignment of each sequencing read to the reference genome in the PyroAlign and Gap Propagation algorithms. Further, we use the standard alignment parameters for SW given in [20] and for ASW given in [13].

Subsection 6.2.1 of this section discusses the experiment design, results and conclusions for objective 1 with respect to each of the distinct MPP read alignment algorithms.

Similarly subsection 6.2.2 discusses the experiment design results and conclusions for objective 2.

6.2.1 Overall Quality Assessment Objective 1

In order to achieve objective 1, we adapt the experiment described in [4], in order to measure the accuracy of an alignment solution produced by an MPP alignment algorithm.

Given a single variant of the reference genome, we generate MPP reads from the variant using PyroSim. We then use an MPP alignment algorithm to produce an alignment solution of the reads.

If the alignment algorithm is a sequence-space algorithm, we obtain a consensus sequence from the alignment solution. We then measure the similarity of this consensus sequence to the original variant sequence.

Our accuracy metric is the percentage measure of the Needleman Wunsch similarity between the consensus sequence and the original variant sequence, divided by the length of the original variant sequence, as shown in Equation 6.1. We refer to the result of Equation 6.1 as the quality of the alignment consensus obtained from employing an MPP read alignment algorithm to MPP reads derived from a single variant of the reference genome.

Equation 6.1:

$$Q = \frac{NW(C,O)}{l} \times 100$$

- Q - Quality of alignment consensus obtained from employing MPP alignment algorithm
- NW(C,0) - Needleman Wunsch similarity between consensus sequence and original variant sequence
- l - Length of the original variant sequence

Flowgram Matching results in a list of flowgrams that have each been individually aligned with respect to the reference genome. If we consider the outputted list of aligned flowgrams to be a multiple alignment of flowgrams for the sake of our experiment, we obtain a mean flow value for each alignment position in the multiple flowgram alignment by averaging the flows (in the aligned MPP flowgrams) that overlap at that position. This gives us a consensus flowgram, which contains a mean flow value for each flow position in the multiple alignment. We subsequently translate the consensus flowgram into a consensus nucleotide sequence by base calling. Given the nucleotide sequence, we can employ Equation 6.1 to measure the Needleman Wunsch similarity between this nucleotide sequence and the original variant sequence.

Note that a list of aligned flowgrams outputted by running Flowgram Matching is not a multiple alignment in the truest sense. We have established Flowgram Matching to be a pairwise alignment algorithm.

As an alternative to Equation 6.1, if the MPP reads are derived from multiple variants as opposed to a single variant sequence, the quality of the alignment consensus following the employment of an alignment algorithm is formulated differently. This formula is provided in subsection 6.2.4.

We measure the accuracy of each MPP alignment algorithm for variants of mutation distance 0.03, 0.05, 0.10 and 0.17 from the reference genome.

As in the previous section, given an MPP read alignment algorithm and a mutation distance from the reference genome, we perform 100 distinct alignments in order to conclusively establish the accuracy of the alignment algorithm for that mutation distance. Each alignment is conducted with a fresh variant (of the same distance from the reference genome) and a fresh run of PyroSim to obtain MPP reads. We also incorporate 15 distinct reference sequences in our experiment.

6.2.2 Results and Discussion

The results for overall quality assessment objective 1 are shown in Figure 6.4.

Given an MPP alignment algorithm, the overall quality of an alignment of all the reads generated from an MPP experiment is directly determined by the quality of the individual pairwise alignments of each of the reads with respect to the reference genome (yielded by the algorithm).

This means that we expect the trend of the overall alignment qualities for SW, ASW, semi-global alignment and Flowgram Matching to mimic the trend of the pairwise alignment qualities for the same algorithms established in subsection 6.1.2. This is evident in Figure 6.4 where the hierarchy of quality for the four algorithms is consistent with the one in subsection 6.1.2.

Recall that we opted for semi-global alignment as the intermediate pairwise alignment strategy for both PyroAlign and Gap Propagation.

In Figure 6.4, we observe that Gap Propagation is a sound algorithm because the quality of the multiple alignments that it produces is consistently better than the overall quality of the corresponding pairwise semi-global alignments that are the intermediate step in Gap Propagation. This establishes the fact that the second step in the algorithm (which is the propagation of gaps through sequencing reads that have each been pairwise

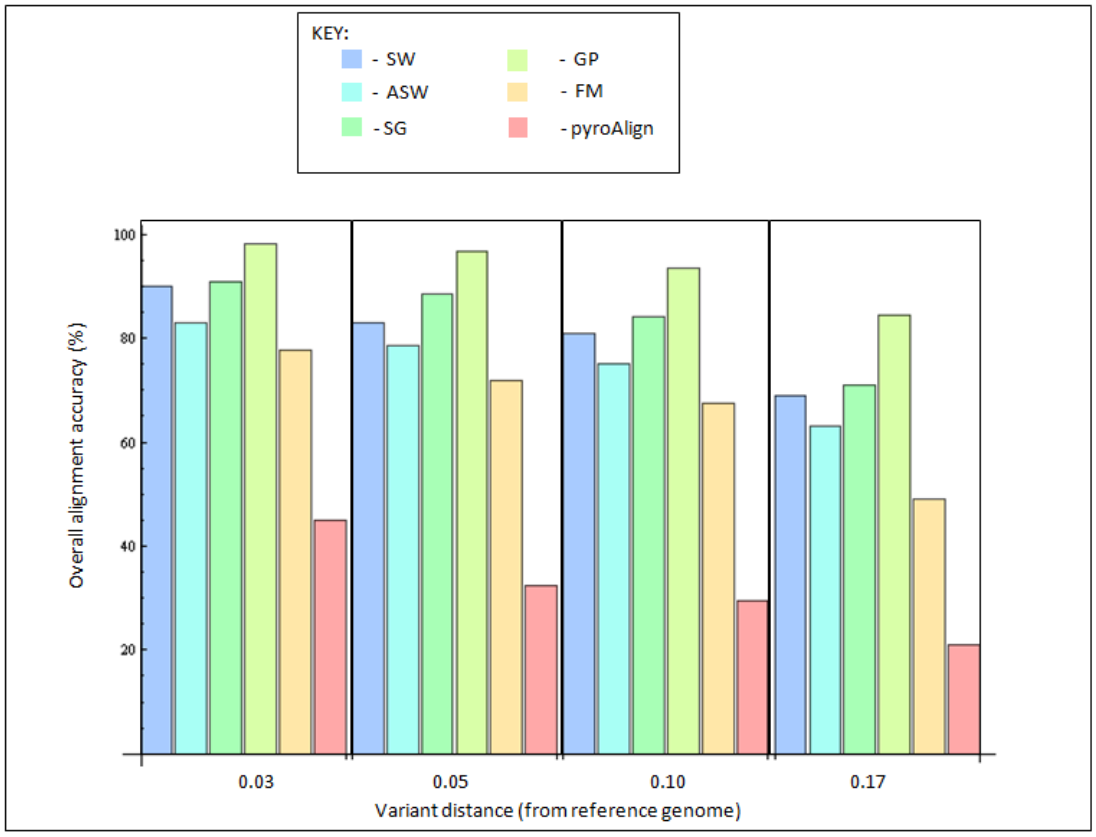

Figure 6.4: Experiment results for overall quality assessment objective 1 - The results show the overall accuracy of the alignment solution produced for the different combinations of MPP read alignment algorithm and variant mutation distance

aligned with respect to the reference genome) serves to improve the overall quality of a multiple alignment of sequencing reads.

This is not the case with PyroAlign, because the quality of the multiple alignment delivered by PyroAlign is poorer than the overall quality of the corresponding pairwise semi-global alignments that are an intermediate step in the PyroAlign algorithm.

In observing the stages involved in PyroAlign multiple sequencing read alignments, we attribute the loss of accuracy to the profile-profile pairwise alignment stage. We established in section 5.4 that this stage of PyroAlign utilizes the clustalw2 program's profile-profile pairwise alignment algorithm. Clustalw2 profile-profile pairwise alignment implements the global pairwise alignment of a pair of input profiles.

A better pairwise profile-profile alignment scheme for PyroAlign would be a localized alignment. This is because after the preceding stages of the algorithm (the alignment of each sequencing read with respect to the reference sequence and the subsequent pairwise alignment of pair of overlapping sequencing reads), the alignment of two profiles to each other should involve just a small tweaking of the profiles with respect to each other.

Let us take the pair of profiles (profile 1 and profile 2) shown in Figure 6.5 as an example. The first two stages of PyroAlign would result in a generalized alignment of the profiles relative to each other as shown in the figure. This generalized alignment subsequently requires a small tweaking in the alignment of profile 1 with respect to profile 2. This tweaking should be done within the alignment window shown in the figure. A clustalw2 profile-profile pairwise alignment of profile 1 and profile 2 is not sensitive enough to the small change that is required because it conducts an alignment on a global scale. The small tweaking required here is a simple insertion of a gap into each sequence in profile 1 after the first nucleotide base of the sequence inside the window.

The result of an inaccurate profile-profile pairwise alignment scheme in the PyroAlign algorithm is a relatively poor performing sequence-space alignment algorithm.

In fact, from our observations the generalized alignment of the sequencing reads with respect to each other achieved by the first two stages of the PyroAlign algorithm is eroded by subsequent clustalw2 pairwise profile-profile alignments.

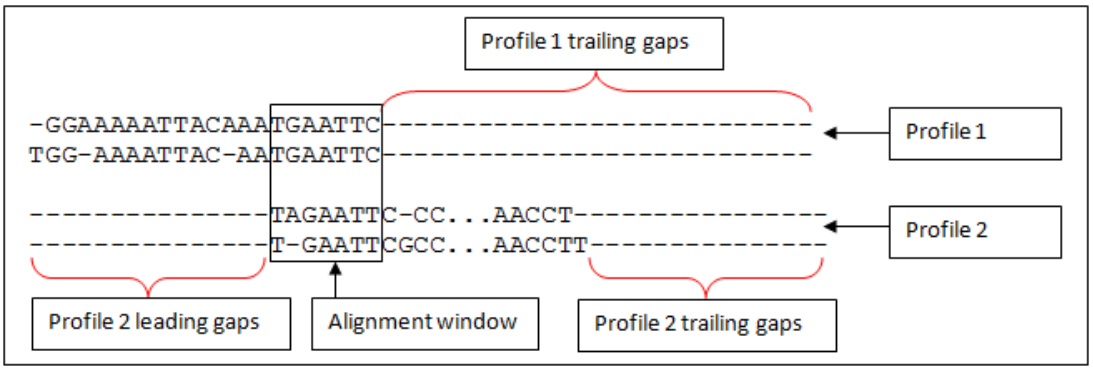

Figure 6.5: A pair of profiles with a generalized alignment relative to each other after stages 1 and 2 of the PyroAlign algorithm - The alignment of p1 to p2 should be done locally within the alignment window shown in the figure as opposed to being conducted global context of the two profiles

6.2.3 Conclusion

We conclude that of the flow-space and sequence-space algorithms compared, Gap Propagation is the most sound and reliable algorithm for the overall placement of MPP reads with respect to the reference genome. PyroAlign, its sequence-space counterpart, is not as reliable.

Flowgram Matching, the only flow-space alignment algorithm implemented here, produces a good overall alignment quality but is outperformed by the sequence-space alignment algorithms.

6.2.4 Overall Quality Assessment Objective 2

Similarly to [4], in order for us to evaluate the ability of an MPP read alignment algorithm to handle reads from multiple variants of the reference genome, we conduct an experiment identical to the one conducted for overall quality assessment objective

1. The key difference between the two experiments is that here the pool of MPP reads used in an experiment run is generated from two variants of the reference sequence (rather than a single variant of the reference sequence).

Given an MPP read alignment algorithm, we measure the algorithm's ability to align a mixture of MPP reads derived from a variants of distance 0.03 and 0.05. The mixture of reads contains equal quantities of reads from each variant sequence.

We also measure the algorithm's ability to align a mixture of MPP reads derived from variants of distance 0.10 and 0.17. Similarly here the mixture of reads contains equal quantities of reads from each variant sequence.

If each read in a mixture of reads is aligned correctly, the distance of the resulting consensus sequence from the reference sequence should estimate the total of the individual distances (from the reference sequence) of the variants from which the reads are derived.

We use a normalized edit distance metric, calculated according to Equation 6.2, to determine the genetic distance between a variant sequence and the reference sequence.

Equation 6.2:

$$NED(v, R) = \frac{ED(v,R)}{Length(R)}$$

- NED - Normalized edit distance between a variant v and the reference sequence R
- ED - Edit distance between a variant v and the reference sequence
- Length(R) - The length of the reference sequence R

Given a list of MPP reads derived from k distinct variants of the reference sequence, the total authentic variation in the reads is obtained by Equation 6.3.

Equation 6.3:

$$TA = \sum_{k=1}^{n} NED(v(k), R)$$

- TA - Total authentic variation of variants (from which MPP reads were derived) from reference sequence
- NED($\boldsymbol{v}(\boldsymbol{k})$,$\boldsymbol{R}$) - Normalized edit distance between variant $\boldsymbol{k}$ ($\boldsymbol{v}(\boldsymbol{k})$) and the reference sequence ($\boldsymbol{R}$)

Therefore given MPP reads derived from multiple variants in an MPP experiment and following the alignment of the reads by an MPP algorithm, the quality of the alignment consensus is given by Equation 6.4.

Equation 6.4:

$$Q = 1 - Abs(TA - NED(C, R))$$

- Q - Quality of alignment consensus produced by MPP alignment algorithm
- TA - Total authentic variation of variants (from which MPP reads were derived) from reference sequence
- NED - Normalized edit distance between alignment consensus (C) and reference sequence (R).

6.2.5 Results and Discussion

The results for overall quality assessment objective 2 are shown in Figure 6.6.

From Figure 6.6, it is evident that the relative accuracies of the MPP read alignment algorithms follow the same trend observed in subsection 6.2.2. This trend is a hierarchy

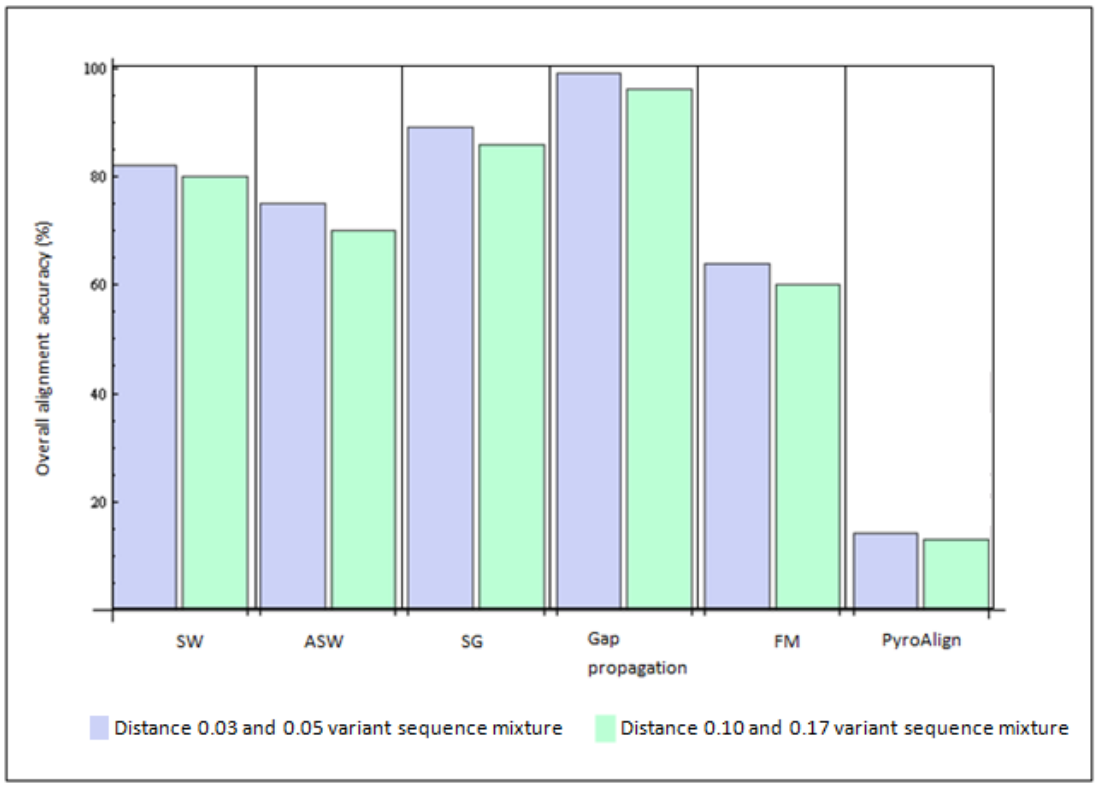

Figure 6.6: Experiment results for objective 2 - The results show overall alignment accuracy of the MPP read alignment algorithms given reads derived from mixtures of variant sequences

of the MPP read alignment algorithms in the following order: Gap Propagation, semi-global alignment, Smith-Waterman alignment, Asymmetric Smith-Waterman alignment, Flowgram Matching, and lastly, PyroAlign.

6.2.6 Conclusion

We conclude that the above hierarchy of the relative qualities of the solutions delivered by the MPP alignment algorithms is true in general.

7

MPP Error Correction - Implementation and Verification

We implemented the following MPP read error correction algorithms:

1. Sequence-space error correction
    - Wang et. al. error correction
    - Eriksson et. al. error correction
2. Flow-space error correction
    - Quince et. al. error correction

Recall that the above error correction algorithms are described in detail in subsection 3.2.2 of this thesis.

All the algorithms are implemented in a Mathematica 8.0 package called MPPErrorCorrect. MPPErrorCorrect can be accessed in two ways:

1. Via the MPPSoft web graphical interface at the URL:

    `http://hammer.cs.ukzn.ac.za/~anisa/`

2. Via the Mathematica 8.0 package, MPPErrorCorrect, which can be downloaded at the URL:

   `http://hammer.cs.ukzn.ac.za/~anisa/MPPErrorCorrect.html`

   .

The complete MPPErrorCorrect user manual is available at the URL:

`http://hammer.cs.ukzn.ac.za/~anisa/MPPErrorCorrectManual.pdf`

This chapter explains how we verified the correctness of our implementations of the algorithms listed above. Further we provide a complexity analysis of the algorithms implemented in MPPErrorCorrect.

7.1 Algorithm Verification

7.1.1 Wang et. al. error correction

We implemented the error correction algorithm described in [13]. The algorithm implementation is straight forward. Given an input multiple sequencing read alignment, Wang error correction should involve the application of Equation 3.13 for each variant nucleotide at a given position in the alignment in order to determine if the nucleotide is authentic. Recall that a nucleotide at a given position in the input multiple alignment is declared to be authentic if Equation 3.13 yields a p-value less than a certain threshold (the threshold is 0.001 in [13]).

However Equation 3.13, which is the same equation provided in [13], is not very clear as it does not provide a starting point for the value $\boldsymbol{k}$ in the summation within the equation. To this end we tried two alternatives:

1. A starting point for $\boldsymbol{k}$ of 0 such that Equation 3.13 is restated as in Equation 7.1.

2. A starting point for **k** of 1 such that Equation 3.13 is restated as Equation 7.2.

Equation 7.1:

$$P = 1 - \sum_{k=\mathbf{0}}^{n-1} \frac{e^{\lambda} \times \lambda^{k}}{k!}$$

Equation 7.2:

$$P = 1 - \sum_{k=\mathbf{1}}^{n-1} \frac{e^{\lambda} \times \lambda^{k}}{k!}$$

Through repeated calculations, we found that the first alternative (Equation 7.1) does not provide us with sensible p-values. Also given the p-values returned, it is not easy to distinguish between significant variation and insignificant variation.

The second alternative (Equation 7.2) excludes the possibly important probability value of $\boldsymbol{k}$=0 from the equation.

We therefore modified Equation 3.10 to obtain Equation 7.3 for the calculation of the p-value of a variant at an alignment position $\boldsymbol{p}$ given that $\boldsymbol{n}$ copies of the variant are observed at $\boldsymbol{p}$ and the total number of reads that overlap at $\boldsymbol{p}$ is $\boldsymbol{N}$.

We find Equation 7.3 is a more reasonable version of Equation 3.13 because given an alignment position $\boldsymbol{p}$, the p-values calculated by the equation will vary accordingly with $\boldsymbol{N}$ the number of reads that overlap at $\boldsymbol{p}$. Equation 7.3 also provides sensible p-values, such that it is easy to distinguish between authentic variants ($P < 0.001$) and non-authentic variants ($P >= 0.001$).

Note that given a multiple sequencing read alignment, Wang et. al. error correction simply pinpoints the significant nucleotide bases at each position of the alignment. Therefore, if the reads in the multiple alignment originate from a single variant of the reference sequence, error correction is inferred by correcting the nucleotides in each read to the most significant nucleotide in the corresponding alignment position.

However given a multiple alignment in which the sequencing reads originate from more than one variant sequence, this method of error correction is not very effective because it does not provide a mechanism of identifying the specific variant sequence that each read originates from. Therefore given a position in the multiple alignment where more than one variant nucleotide is found to be significant, the algorithm falls short of identifying which variant nucleotide is correct with respect to a read in the alignment.

Equation 7.3:

$$P = \sum_{k=n}^{N} \frac{e^{\lambda} \times \lambda^{k}}{k!}$$

- P - Probability that specific variant would occur $\boldsymbol{n}$ or more times if it were a sequencing error.
- n - Number of occurrences of specific variant at alignment position $\boldsymbol{p}$
- N - Total number of aligned sequencing reads that overlap at alignment position $\boldsymbol{p}$
- μ - Mean error rate per nucleotide position (Recall $\mu = 0.0007$ in non-homopolymeric regions and $\mu = 0.0044$ in homopolymeric regions)
- λ - Expected number of errors. This is calculated by $\lambda = N \times \mu$

We now seek to verify the correctness of our implementation of this method of error correction by establishing our algorithm to be a sound algorithm for the correction of errors in reads originating from a single variant of the reference genome. We do this conclusively by employing the following argument.

- In theory, an error correction algorithm should remove all the non-authentic variation from each read in a list of MPP reads. In sequence-space, this means that following error correction, each sequencing read should be identical to the original fragment from which it was derived.

  If we allow for the fact that the error correction algorithm may not be perfect, the genetic similarity of a sequencing read to its corresponding original fragment following error correction should at least be higher than the similarity of the read to the original fragment prior to error correction.

Recall that a PyroSim simulation accompanies each outputted MPP read with the corresponding original fragment from which the read was derived.

Note that the above argument only applies if a high quality multiple alignment is input into the Wang error correction algorithm. The input multiple alignment has to be sensible in order for the algorithm to be able to detect sequencing errors because error correction is conducted independently at each position of the alignment. This requires the sequencing reads in the multiple alignment to have been aligned accurately such that the algorithm has good estimates for distributions of bases at each position of the alignment. Recall that in the previous chapter of this thesis, we established Gap Propagation as the superior sequence-space alignment algorithm.

The ensuing subsections discuss an experiment to establish our implementation of the Wang et. al. algorithm to be sound.

7.1.1.1 Wang error correction experiment

We say that a sequence-space error correction method is sound if we can establish that overall the genetic similarity between the error corrected aligned reads and the corresponding fragments from which they were derived is higher than the genetic similarity between the aligned reads and the corresponding fragments prior to error correction.

To this end we set up an experiment by deriving sequencing reads from a single variant of the reference sequence. Subsequently we use Gap Propagation to obtain a high quality multiple sequencing read alignment. For the sake of our experiment, the corresponding original fragment for each sequencing read is also incorporated into the multiple alignment.

Equation 7.4:

$$A_1 = \frac{1}{N} \sum_{j=1}^{N} \left(\frac{NW(r_j, o_j)}{Length(o_j)} \times 100 \right)$$

- A_1 - Average normalized Needleman Wunsch similarity between aligned sequencing read vector $r_{1...N}$ and aligned original fragment vector $o_{1...N}$
- N - Total number of aligned sequencing reads
- r_j - The j^{th} aligned sequencing read
- o_j - The j^{th} aligned original fragment
- $NW(r_j, o_j)$ - Needleman Wunsch similarity between aligned sequencing read r_j and the corresponding aligned original fragment o_j
- $Length(o_j)$ - Length of the j^{th} aligned original fragment o_j

Equation 7.5:

$$A_2 \;=\; \frac{1}{N}\sum_{j=1}^{N}\left(\frac{NW(c_j, o_j)}{Length(o_j)} \times 100\right)$$

- A_2 - Average normalized Needleman Wunsch similarity between error corrected aligned sequencing read vector $c_{1...N}$ and aligned original fragment vector $o_{1...N}$
- N - Total number of aligned sequencing reads
- c_j - The j^{th} error corrected aligned sequencing read
- o_j - The j^{th} aligned original fragment
- $NW(c_j, o_j)$ - Needleman Wunsch similarity between error corrected aligned sequencing read c_j and the corresponding aligned original fragment o_j
- $Length(o_j)$ - Length of the j^{th} aligned original fragment o_j

Given $\boldsymbol{N}$ aligned sequencing reads ($r_{1...N}$) and $\boldsymbol{N}$ corresponding aligned original fragments ($o_{1...N}$) such that the j^{th} aligned original fragment is obtained by incorporating the fragment from which the j^{th} aligned sequencing read originates into the multiple alignment of the reads, we calculate the average genetic similarity between $r_{1...N}$ and $o_{1...N}$ according to Equation 7.4.

We now apply our Wang et. al. error correction algorithm to the multiple sequencing read alignment. This is such that we derive a corrected aligned read c_j for each sequencing read r_j in the vector $r_{1...N}$. Following this we calculate the average genetic similarity between the vector of error corrected aligned sequencing reads, $c_{1...N}$, and the vector of corresponding aligned original fragments $o_{1...N}$ according to Equation 7.5.

Our theory states that if our Wang error correction algorithm is sound, A_2 calculated according to Equation 7.5 should be greater than A_1 calculated according to Equation 7.4.

We conduct distinct experiments for variants of mutation distance 0.03, 0.05, 0.10 and 0.17 from a reference genome sequence. We use reference genome sequences and variant sequences of length 150 bp. For each experiment, the sequencing reads are derived from a population of 50 copies of a single variant sequence. Given a specific mutation distance, we perform 100 runs of the experiment. Each experiment run is conducted with a fresh variant (of the same distance from the reference genome) and a fresh run of PyroSim to obtain sequencing reads. We also incorporate 15 distinct reference sequences in our experiment.

7.1.1.2 Results and Discussion

Figure 7.1 displays the results for Wang algorithm verification objective 1. Specifically, given a variant distance from the reference sequence, the results show the average value of the difference between A_2 and A_1 (i.e. $A_2 - A_1$) for the specific variant distance. We refer to this difference as the average error correction margin.

We note that the average error correction margin is always positive (i.e. $A_2 \geq A_1$) for the distinct variant distances. This indicates that on average Wang et. al. error correction improves the similarity of an aligned sequencing read to the original fragment from which the read was derived.

During the conduction of our experiment runs we noted that Wang et. al. error correction will improve the accuracy of the majority (above 85%) of the reads in the experiment. The small number of aligned reads for which this was not the case had not been aligned correctly with respect to the reference genome by the alignment algorithm preceding error correction. Error correction on a misaligned read is likely to be inaccurate.

The rate at which a PyroSim simulation confers errors onto sequencing reads depends on two factors; the simulation FVDM and the lengths of the stretches of homopolymers in the reads. This means that the MPP error rate does not vary significantly as we observe different variant distances from the reference genome. By extension, the average value of A_1 calculated according to Equation 7.4 does not vary significantly with different variant distances from the reference genome.

On the other hand, the value of A_2 depends on the quality of the error correction algorithm, which in turn depends on the quality of the alignment algorithm that precedes error correction. Recall that experiment results derived in the previous chapter of this thesis established that in general the accuracy of an alignment algorithm declines with increasing authentic variation in the sequencing reads that are input into the algorithm. This is because a sequencing read derived from a variant that has a high genetic distance from the reference sequence is more difficult to align with respect to the reference sequence.

Overall we therefore expect that the accuracy of error correction should decline with increasing variation from the reference genome. This is reflected in Figure 7.1 whereby despite consistency in the A_1 values across variant mutation distances, there is a decline in the A_2 values. This is reflected in a decline in the error correction margin with increasing mutation distance from the reference genome.

7.1.1.3 Conclusion

We conclude that our implementation of the Wang et. al. algorithm is a sound algorithm for correcting non-authentic variation in sequencing reads derived from a single variant of the reference genome.

7.1.2 Eriksson et. al. error correction

We implemented the algorithm described in [3].

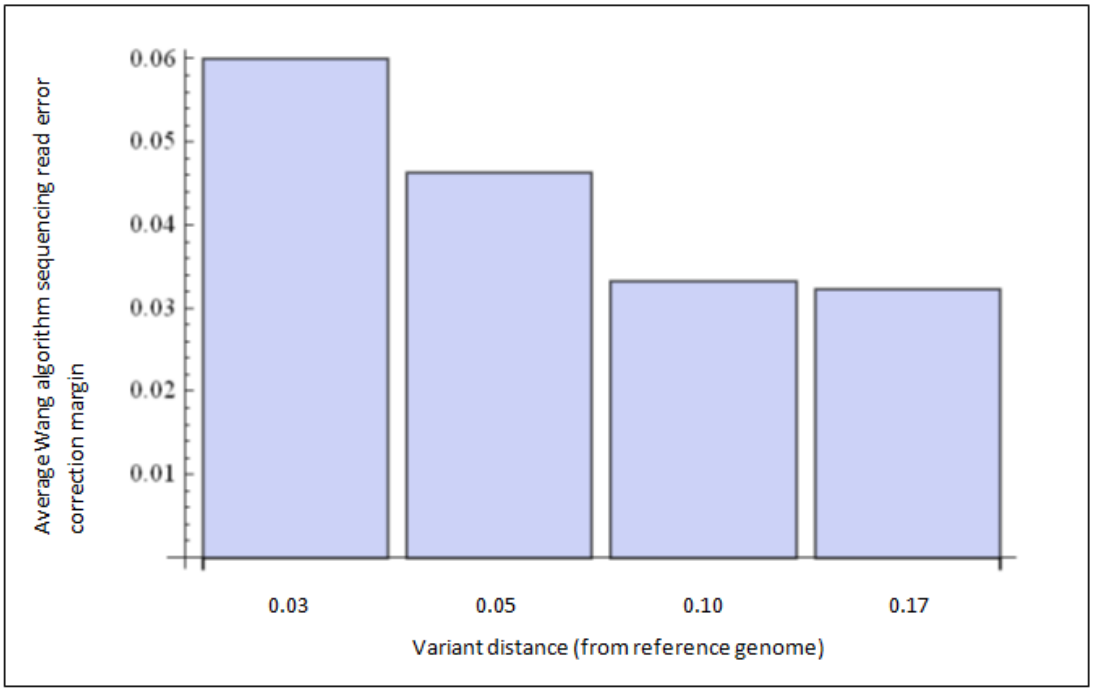

Figure 7.1: Experiment results for Wang algorithm verification - The results show the average error correction margin for variants of mutation distance 0.03, 0.05, 0.10 and 0.17 from the reference genome

Recall that the Eriksson et. al. error correction algorithm is more elaborate than Wang et. al. error correction. This is because the algorithm attempts to count the number of variants underlying the reads in the input multiple alignment. Subsequently the algorithm attempts to cluster the aligned sequencing reads about the sequences from which they originate. Each sequencing read is then corrected to the consensus sequence of the cluster in which it is placed. This procedure is conducted over several overlapping windows of the multiple alignment.

It follows that Eriksson et. al. error correction can be used to correct sequencing reads that originate from multiple variants of the reference genome.

In order to verify the soundness of our implementation of Eriksson error correction, we conduct an experiment identical to the one described in subsection 7.1.1.

Further we repeat the experiment for sequencing reads derived from mixtures of variants of the reference genome. Specifically we derive sequencing reads from a mixture of two distinct variants of distance 0.03 and 0.05 from the reference genome as well as a mixture of two distinct variants of distance 0.10 and 0.17 from the reference genome.

7.1.2.1 Results and Discussion

The results derived from conducting the experiment with single variants of the reference genome are shown in Figure 7.2.

Similarly to the results for the Wang et. al. algorithm, in Figure 7.2 we observe that the average error correction margin is consistently positive. Also we observe that the quality of error correction declines with an increase in variation from the reference sequence.

A comparison of Figure 7.2 to Figure 7.1 shows that our implementation of Eriksson error correction produces results that are slightly better than the results produced by our implementation of Wang error correction. We attribute this to a hypothesis that we formulated in subsection 3.2.2.1 of this thesis. Here we stated that the Eriksson metric

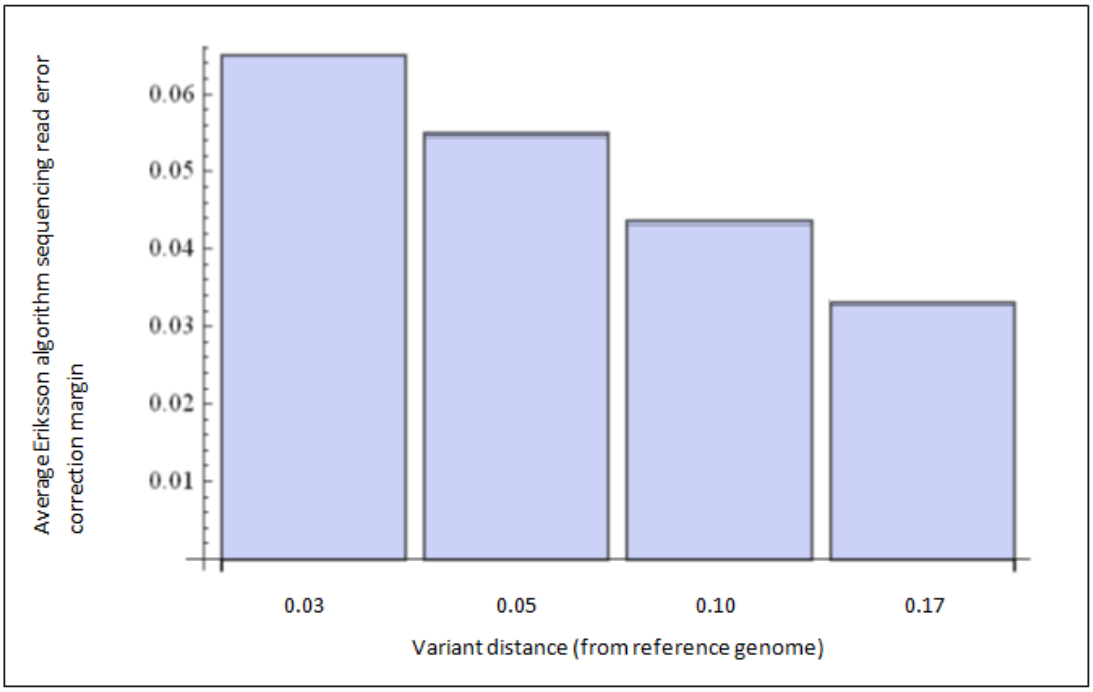

Figure 7.2: Experiment results for Eriksson algorithm verification - The results show the average error correction margin for variants of mutation distance 0.03, 0.05, 0.10 and 0.17 from the reference genome

for the significance of a variant at a given alignment position is potentially better than the Wang metric for the same because of the finer granularity applied by the Eriksson algorithm.

Note that as part of its functionality, our Eriksson algorithm deletes duplicate reads in the input multiple alignment. As a result, the reads that are corrected by the algorithm are a subset of the reads constituting the input multiple alignment.

On the other hand, our Wang algorithm returns corrections for all the reads in the input multiple alignment.

Therefore if we input the same multiple sequencing read alignment into both error correction algorithms, the superior performance of the Eriksson algorithm may not be evident in the overall average A_2 scores.

However on closer observation we see that given a sequencing read in the input multiple alignment, the error correction results on the read yielded by the Eriksson algorithm are either better than or in most cases equivalent to the error correction results on the read yielded by the Wang algorithm.

The results derived from conducting the experiment on sequencing reads derived from mixtures of variants of the reference genome are shown in Figure 7.3.

It is evident from Figure 7.3 that our implementation of Eriksson error correction is sound for reads derived from mixtures of distinct variants from the reference genome.

7.1.2.2 Conclusion

We conclude that our implementation of Eriksson et. al. error correction is a sound algorithm for the correction of MPP errors in sequencing reads derived from a single variant from the reference genome. Further the algorithm can also be used to correct reads derived from multiple variants of the reference genome.

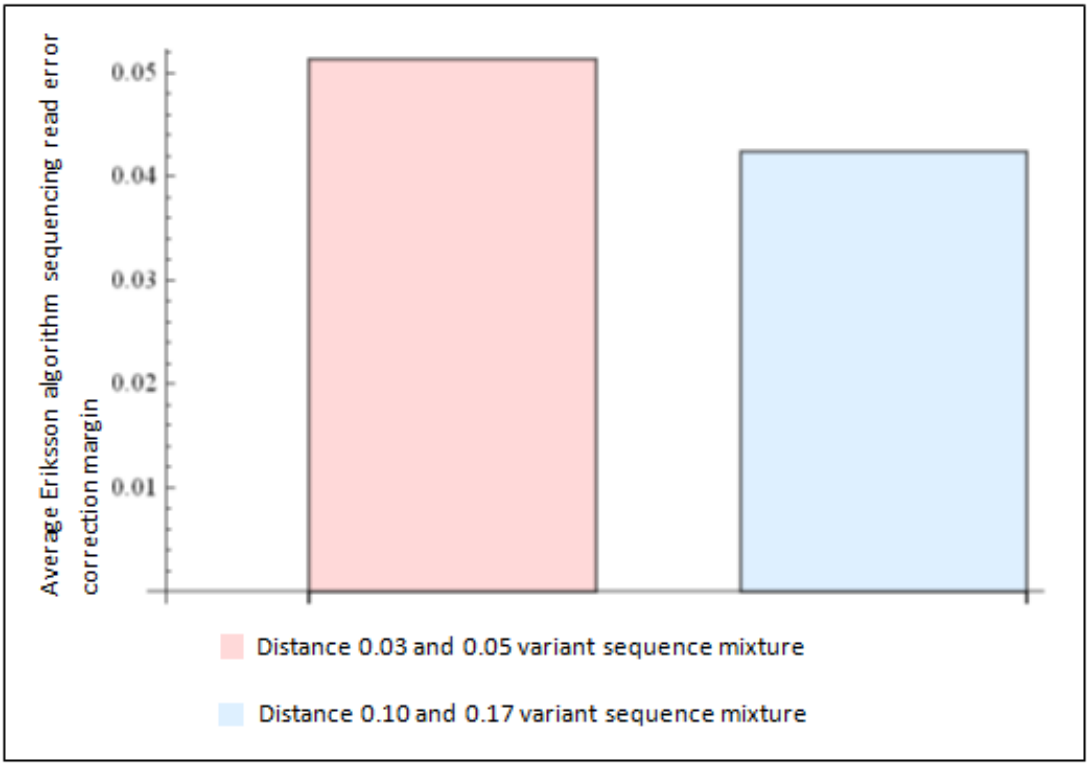

Figure 7.3: Experiment results for Eriksson algorithm verification - The results show the average error correction margin for mixtures of variants of distance 0.03 and 0.05 from the reference sequence as well as mixtures of variants 0.10 and 0.17 from the reference sequence

7.1.3 Quince et. al. error correction

We implemented the algorithm described in [15]. Recall that this is a flow-space error correction algorithm. Further, the algorithm implements expectation maximization, given a set of aligned flowgrams, in order to determine the variant sequences underlying the flowgrams (refer to subsection 3.2.2 of this thesis).

We use Flowgram Matching for alignment of flowgrams to flow-space representations of underlying sequences in our version of Quince et. al. error correction.

We implement a maximum of 1000 iterations of the EM algorithm. However, we terminate the algorithm if it converges to a local maximum before this number of iterations is reached.

In order to prevent our version of the Quince EM algorithm from converging to a solution too soon, we introduce some minor random variation into the underlying sequences calculated (according to Equation 3.15) for every fixed number of iterations of the algorithm. We do this up to 50 iterations of the algorithm.

Further, in order to guide the convergence of the EM, we replace the alignments of some flowgrams with respect to the calculated underlying sequences by the original alignments of the same flowgrams with respect to the reference sequence for every fixed number of iterations of the algorithm. As above, we do this up to 50 iterations of the algorithm. This additional step in our version of the algorithm assumes that the original alignments of the flowgrams with respect to the reference sequence are accurate.

After a burn in phase of 60 iterations of the EM, we surmise that the algorithm has converged if there is no change in the calculated sequences underlying each cluster for a contiguous 20 iterations of the EM.

Recall that in subsection 6.1.2, we established that Flowgram Matching may mis-align a small fraction of individual flowgrams with respect to the original sequence.

In order to minimize the effects of flowgram mis-alignment, our implementation of this method of error correction requires more coverage than its sequence space error cor-

rection counterparts. This is such that the relatively large number of input flowgrams is more likely to average out the effects of the small fraction of mis-aligned flowgrams.

We now seek to establish our implementation of Quince error correction as a sound flow-space error correction algorithm.

Because Flowgram Matching has a higher tendency to mis-align MPP reads than its sequence-space alignment counterparts, we opt not to analyze the quality of error correction on the basis of the similarity between each corrected read and its corresponding original fragment (as we did with Wang and Eriksson error correction). This is because each mis-aligned read will be corrected to the consensus of its new alignment, leading to a very low similarity between the read and the original fragment from which it was derived.

Instead we look at the results of this method of error correction holistically. We analyze the similarity between the sequences that the algorithm estimates to underlie the input flowgrams and the true authentic variant sequences from which the flowgrams originate.

7.1.3.1 Quince error correction verification experiments

- **Experiment 1**

  Given a single variant of the reference genome, we derive MPP flowgrams from the variant. Subsequently we input the raw flowgrams into our error correction algorithm, specifying that the cut-off distance for the preliminary complete linkage clustering algorithm (which counts the number of clusters underlying the flowgrams - Refer to subsection 3.2.2.2) is 1.0. This counts the number of clusters underlying the flowgrams as 1. This means that the algorithm will try to derive the single sequence underlying the flowgrams.

  Note that in order to minimize flowgram mis-alignment, we employ longer reads for this method of error correction. We set the mean fragment length for MPP

simulation to 30.

Subsequently, we determine the normalized Needleman-Wunsch similarity between the underlying sequence calculated by the error correction algorithm and the original variant sequence from which the flowgrams were derived. We repeat this procedure for authentic variants of distance 0.03, 0.05, 0.10 and 0.17 from the reference genome.

For each experiment run, whereby a list of flowgrams is input into the error correction algorithm, we design a control by running Flowgram Matching on the list of flowgrams and subsequently deriving a consensus flowgram as described in subsection 6.2.1. The consensus flowgram is translated into a consensus sequence by base calling. We therefore compare the normalized Needleman-Wunsch similarity between the consensus sequence and the authentic variant sequence to the normalized Needleman-Wunsch similarity between the underlying sequence calculated by the error correction algorithm and the authentic variant sequence. If the latter is found to be consistently larger than the former, our error correction method is validated.

Recall that Quince error correction requires a relatively large MPP read coverage. We perform each experiment with reads derived from 100 copies of the single variant sequence. Given a specific mutation distance, we perform 100 runs of the experiment. Each experiment run is conducted with a fresh variant (of the same distance from the reference genome) and a fresh run of PyroSim to obtain sequencing reads. We also incorporate 15 distinct reference sequences in our experiment.

- **Experiment 2**

  We now repeat experiment 1 described above for MPP flowgrams derived from mixtures of variants of the reference genome sequence. Specifically we derive

flowgrams from a mixture of two distinct variants of distance 0.03 and 0.05 from the reference genome as well as a mixture of two distinct variants of distance 0.10 and 0.17 from the reference genome. In each case we select a cut-off distance for the complete linkage clustering algorithm that will count the number of clusters underlying the flowgrams as 2. The error correction algorithm will therefore estimate the genetic composition of two sequences underlying the flowgrams.

In this case, given an experiment run whereby a list of flowgrams is input into the error correction algorithm, we derive an experiment control by first using flowgram matching to align each of the flowgrams in each cluster with respect to the reference genome. Subsequently, we clustering the flowgrams (using complete linkage clustering - the preliminary algorithm in this method of error correction) according to their alignments with respect to the reference sequence. We then derive a consensus flowgram for each cluster (as in subsection 6.2.1) from which we obtain a consensus sequence for each cluster.

We compare the average normalized Needleman-Wunsch similarity between the consensus sequences derived from the steps above and the original variant sequences with the average normalized Needleman-Wunsch similarity between the underlying sequences calculated by the error correction algorithm and the original variant sequences. If the latter is found to be consistently larger than the former, our error correction method is validated.

- **Results and Discussion**

Figures 7.4 and 7.5 show the results for experiment 1 and experiment 2 respectively. We observe that in both cases, the results derived by the Quince algorithm are better than the control results. Further, the mean Needleman Wunsch metric pertaining to the Quince algorithm results is consistently above 60% in both cases. This implies that given a list of input flowgrams, our version of Quince

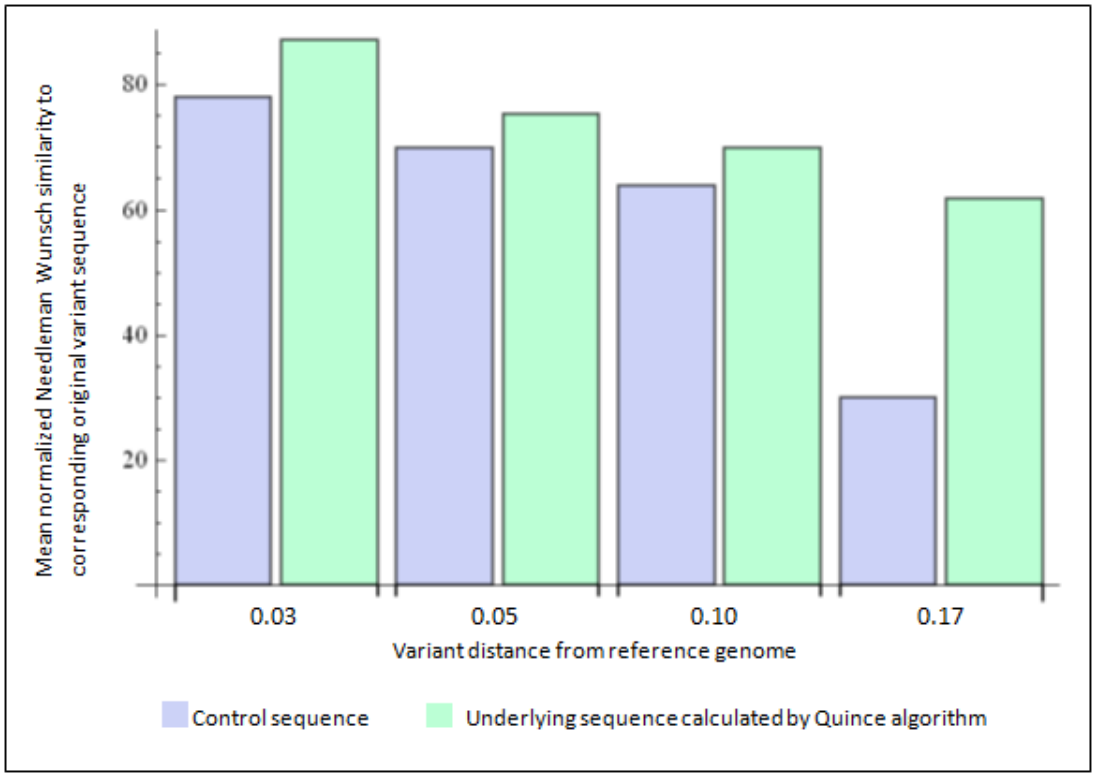

Figure 7.4: Experiment results for Quince algorithm verification experiment 1 - Given a variant of a specific mutation distance from the reference genome sequence, the results show a comparison of
1. The average normalized Needleman Wunsch similarity between the control sequence and the original variant sequence
And
2. The average normalized Needleman Wunsch similarity between the underlying sequence calculated by the Quince algorithm and the original variant sequence

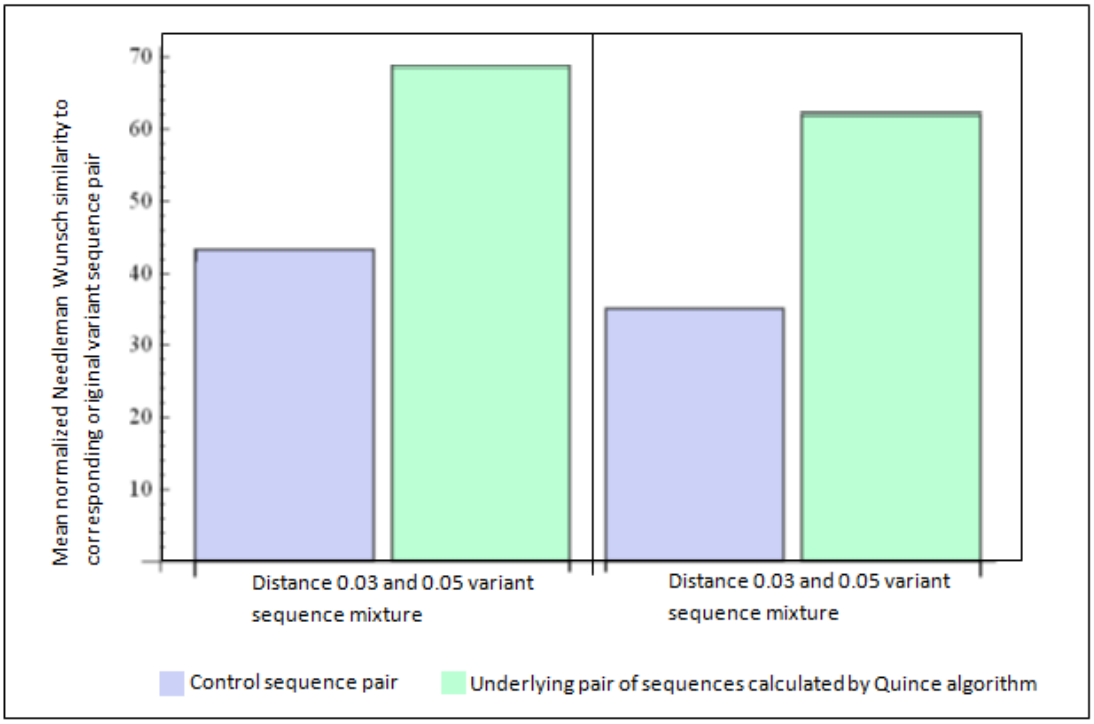

Figure 7.5: Experiment results for Quince algorithm verification experiment 2
- Given a mixture of variant sequences, the results show a comparison of
1. The average normalized Needleman Wunsch similarity between the control sequence pair and the original variant sequence pair
And
2. The average normalized Needleman Wunsch similarity between the pair of underlying sequence calculated by the Quince algorithm and the original variant sequence pair

et. al. error correction is able to reconstruct up to 60% of each variant sequence underlying the flowgrams. Recall this is under specialized conditions of better coverage as well as MPP flowgrams derived from longer template DNA fragments than the sequence-space error correction algorithms.

- **Conclusion**

  We conclude that our implementation of the Quince et. al. algorithm is a sound flow-space error correction algorithm.

7.2 Complexity Analysis

From Appendix C1, we estimate the complexity of Wang error correction to be $\Theta(N + L_\Gamma)$, where N and L_Γ are the number of aligned sequencing reads input into the algorithm and the length of the reference genome respectively. This arises from the computations performed on each aligned sequencing read by the algorithm prior to actual error correction, followed by an error correction phase that corrects each column of the input multiple alignment of sequencing reads. There are approximately L_Γ columns in the multiple alignment.

We estimate the overall complexity of our implementation of Eriksson error correction to be $\Theta(N + NL_\Gamma)$ (Refer to Appendix C2).

From Appendix C3, the complete linkage clustering phase of Quince error correction is of complexity $\Theta(N^2)$. Subsequently, each iteration of the algorithm is of complexity $\Theta(N^2+NL_\Gamma+NL_\Gamma L_R)$ (Recall that L_R is the average template DNA fragment length). We therefore see that the complexity of a single iteration of the Quince algorithm is higher than the overall complexity for both the Wang algorithm and the Eriksson algorithm. Further, recall that the Quince algorithm may require up to 1000 iterations to converge.

Figures 7.6 and 7.7 display the variation in execution time for the Wang and Eriks-

son algorithms with different parameterizations.

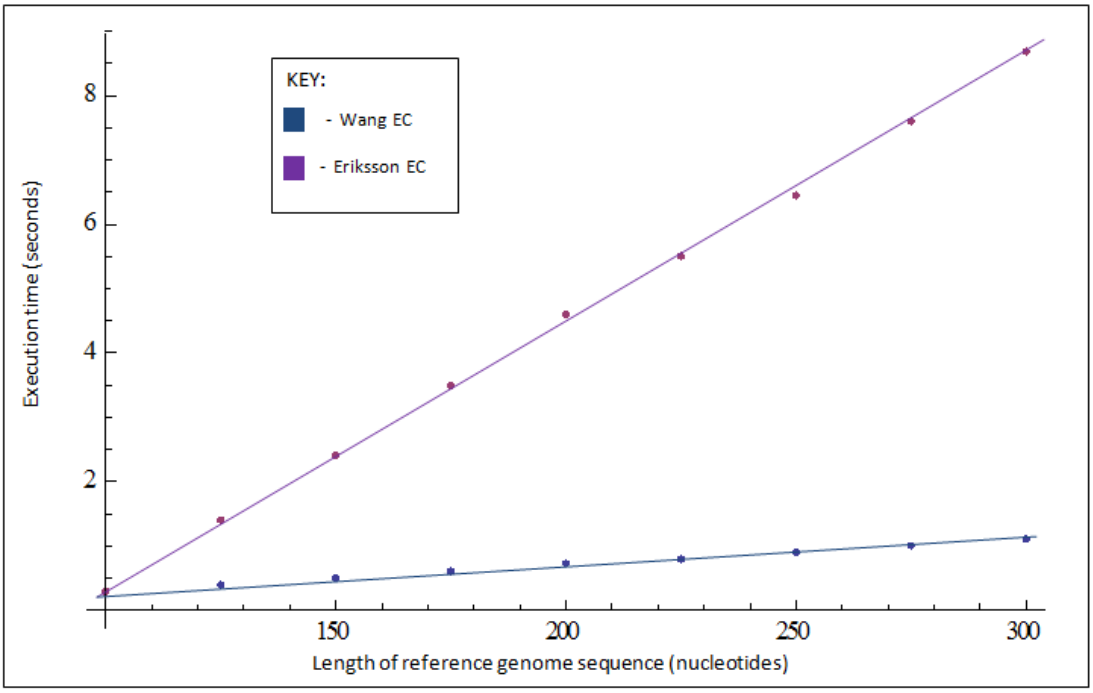

Figure 7.6: Dependence of error correction algorithm execution time on reference genome nucleotide sequence length - As predicted by the complexities, the curves for both Wang and Eriksson error correction are linear. We observe a steeper gradient for Eriksson EC, also predicted by the complexities

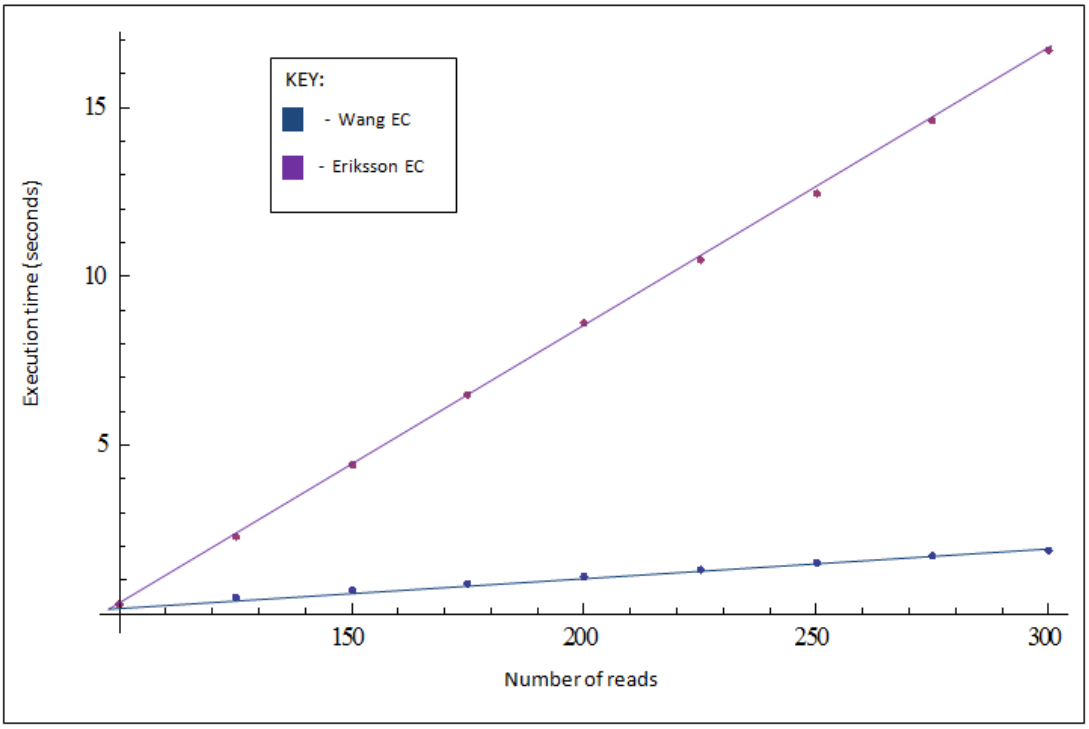

Figure 7.7: Dependence of error correction algorithm execution time on the number of sequencing reads - As predicted by the complexities, the curves for both error correction are linear. We observe a steeper gradient for Eriksson EC, also predicted by the complexities

8

MPP Error Correction - Qualitative Comparison

In the previous chapter we established Eriksson et. al. error correction to be a suitable representative for sequence-space error correction given a multiple alignment of sequencing reads that originate from multiple variants of the reference genome.

Similarly, we concluded that Quince et. al. error correction is a suitable representative for flow-space error correction.

We now compare the two algorithms in order to determine the algorithm that yields the best qualitative results. We do this bearing in mind the argument in literature that postulates that flow-space is a better domain for obtaining accurate results from MPP reads than sequence-space, because of valuable information lost by base calling[7, 15].

8.1 Error correction Qualitative Comparison Experiment

We conduct the comparison of the Eriksson and Quince error correction methodologies by running a simple experiment.

In this experiment we compare the ability of the algorithms to reconstruct under-

lying variant sequences from MPP reads as the number of underling variant sequences increases.

Given a set of MPP flowgrams from an experiment we run Quince error correction on the flowgrams in order to determine the sequences underlying the flowgrams. We then perform base calling on the flowgrams to obtain sequencing reads and subsequently derive a multiple alignment of the sequencing reads using Gap Propagation. We input the multiple alignment into the Eriksson error correction algorithm to obtain an error corrected multiple alignment. Subsequently, we determine the consensus nucleotide sequences of the error corrected multiple alignment.

Given this, we determine the mean normalized Needleman Wunsch similarity between the sequences underlying the flowgrams as computed by the Quince algorithm and the true variants underling the flowgrams. We compare this metric to the mean normalized Needleman Wunsch similarity between the consensus nucleotide sequences derived from the Eriksson error corrected multiple sequencing read alignment and the true variants underlying the flowgrams.

As a control we determine the consensus sequences of the multiple sequencing alignment prior to Eriksson error correction and determine the mean normalized Needleman Wunsch similarity between these consensus nucleotide sequences from the uncorrected alignment and the true variants underlying the flowgrams.

We conduct this experiment for a total number of underlying variants ranging from 1 to 6.

In each experiment run, the number of copies of each underlying variant is 100. We derive the underlying variants from a reference sequence of length 150 bp. The mean fragment length for MPP simulation is set at 30 with a standard deviation of 6.5 (Recall that Quince error correction requires longer MPP reads in order to minimize flowgram mis-orientation). For each number of distinct underlying variants we perform 100 experiments to obtain suitable average scores.

Each experiment run is conducted with a fresh variant (of the same distance from the reference genome) and a fresh run of PyroSim to obtain sequencing reads. We also incorporate 15 distinct reference sequences in our experiment.

8.1.1 Results and Discussion

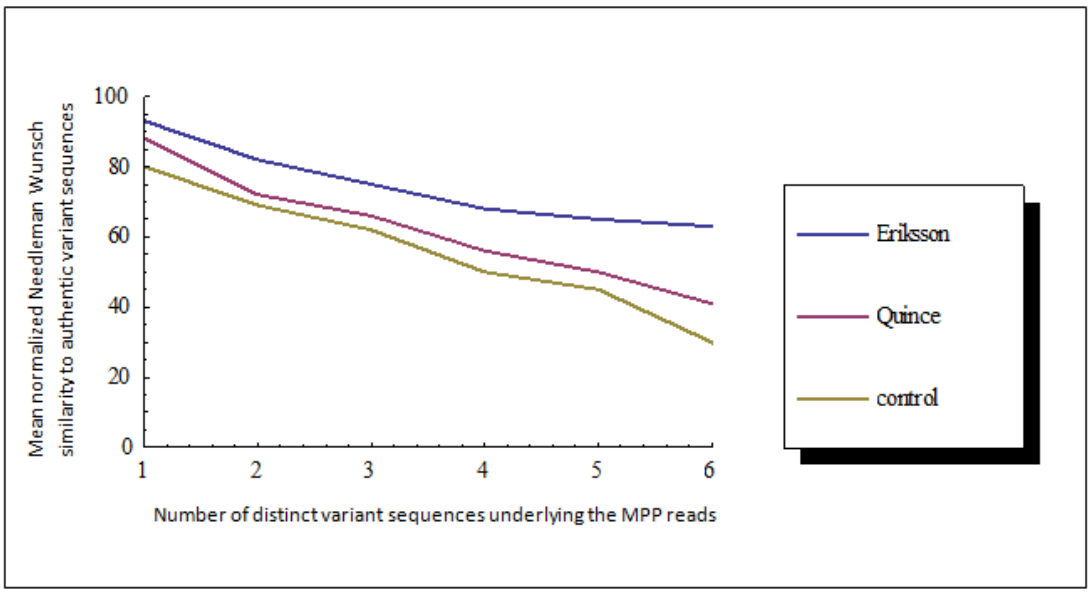

Figure 8.1: Error correction qualitative analysis results - The results show that Eriksson error correction consistently outperforms Quince error correction

The results for our qualitative analysis are provided in Figure 8.1. Here we see that Eriksson error correction is a better algorithm than Quince error correction as the former consistently outperforms the latter as the number of underlying variant sequences varies.

We attribute this discrepancy to base calling. Recall that base calling converts flowgrams (the raw data for the Quince algorithm) into sequencing reads (the raw data for the Eriksson algorithm). Further, recall that in subsection 6.1.2, we established that base calling leads to better MPP read alignments because it eliminates non-authentic flows that occur at the tail ends of reads. Since MPP read alignment precedes error

correction, the quality of the latter is highly dependent on the quality of the former. Despite the fact that the Quince algorithm may be sound, the result of Quince error correction is highly likely to be flawed by an unreliable alignment algorithm.

Also in Figure 8.1, we observe that the gradient of the decline in the quality of the error corrected solution yielded by the Quince algorithm is steeper than the same for the Eriksson algorithm. This indicates that the results yielded by the Quince algorithm are not reliable for a large number of underlying variants.

8.1.2 Conclusion

We conclude that in this scenario, the sequence-space algorithm outperforms the flow-space algorithm.

Note that overall a flow-space algorithm will certainly always perform poorly if the input MPP reads are derived from an experiment with an excess number of flow cycles. In sequence-space we credit base calling for removing the effect of excess flows from derived sequencing reads.

9

Conclusion

This study aimed to make a qualitative comparison of distinct MPP read alignment and error correction algorithms.

Our qualitative comparison was facilitated by the design of an MPP simulator, PyroSim. The purpose of our simulator was:

1. Given an MPP read alignment algorithm, to enable the comparison of the alignment of MPP reads yielded by the algorithm to the authentic alignment of the reads

2. Given an MPP error correction algorithm, to enable the comparison of the sequences of nucleic acids in MPP reads following error correction to the authentic sequences of nucleic acids in the template fragments from which the MPP reads were derived

We implemented three complete software packages:

- PyroSim - An MPP simulation package
- MPPAlign - An MPP read alignment package
- MPPErrorCorrect - An MPP read error correction package

Further, we implemented MPPSoft, a public interface that facilitates MPP simulation, alignment and error correction. The interface can be accessed at the public URL:

`http://hammer.cs.ukzn.ac.za/~anisa/`

The qualitative comparison of the alignment algorithms implemented in MPPAlign found Gap Propagation to be the best performing alignment algorithm.

Further, we discovered that the flow-space algorithm, Flowgram Matching, is very sensitive to the number of flow cycles set for MPP simulation in the experiment that derives the input MPP reads. Also, Flowgram Matching is sensitive to the degree of variation of the input flowgrams with respect to a reference sequence. Due to this sensitivity, Flowgram Matching is not as reliable as its sequence-space alignment counterparts. However, we further discovered that Flowgram Matching is the most scalable algorithm and consistently yields the lowest execution times.

The pairwise alignment algorithms (Smith Waterman and Asymmetric Smith Waterman alignment, as well as Flowgram Matching) are found to yield equivalent computational complexities. Gap Propagation has a higher complexity than the pairwise alignment algorithms, but this is offset by a better alignment quality resulting from the additional step in the algorithm.

Overall, Smith Waterman and Asymmetric Smith Waterman alignment, Gap Propagation and Flowgram Matching are scalable and do not require parallelization. PyroAlign, however, requires parallelization in order to be scalable.

We conclude the following qualitative hierarchy for the solutions delivered by the MPP alignment algorithms implemented in the MPPAlign package:

1. Gap Propagation

2. Semi-Global alignment

3. Smith Waterman alignment

4. Assymetric Smith Waterman alignment

5. Flowgram Matching

6. PyroAlign

The qualitative comparison of the error correction algorithms implemented in MPPErrorCorrect found the Eriksson algorithm to be the best performing error correction algorithm. Although the Eriksson algorithm has a higher complexity than Wang error correction, the trade off is a more accurate alignment solution and the ability of the algorithm to cater for sequencing reads derived from multiple variants of the reference genome.

The sensitivity of working in flow-space lead to the relatively poor performance of Quince error correction.

This leads us to surmise that flow-space is a sensitive domain. Indeed it is likely that a sequence-space algorithm will outperform a flow-space algorithm. However, taking quality control measures with the flowgrams input into a flow-space algorithm can lead to better results for flow-space computations. An example of a quality control measure is restricting the number of flow-cycles in the MPP experiment from which the flowgrams are derived.

We conclude the following qualitative hierarchies for the solutions delivered by the MPP error correction algorithms implemented in the MPPErrorCorrect package:

- For the correction of MPP reads derived from more than one variant of the reference genome:

    1. Eriksson error correction

    2. Quince error correction

- For the correction of MPP reads derived from a single variant of the reference genome:

    1. Eriksson error correction
    2. Wang error correction
    3. Quince error correction

References

[1] Huse S., Morrison H.G., Sogin M.L., Welch D.M., Huber J.A. **Accuracy and quality of massively parallel DNA pyrosequencing**. *Genome Biology* **8**, R143, (2007). Available http://genomebiology.com/2007/8/7/R143. 2, 18, 72

[2] Zagordi O., Roth V., Beerenwinkel N., Geyrhofer L. **Deep sequencing of a Genetically Heterogeneous Sample: Local Haplotype Reconstruction and Read Error Correction**. *Journal of Computational Biology*, **17** (3):417-28 (2009). Available http://www.ncbi.nlm.nih.gov/pubmed/20377454. 2, 4, 65, 66

[3] Eriksson N., Mitsuya Y., Rhee S., Wang C., Gharizadeh B., Ronaghi M., Shafer R., Beerenwinkel N., Pachter L. **Viral Population Estimation Using Pyrosequencing**. *PLOS Computational Biology*, **4**(5): e1000074. (2008). Available http://www.ncbi.nlm.nih.gov/pubmed/18437230. 2, 3, 4, 24, 44, 61, 107, 155

[4] Saeed F., Zagordoi O., Beerenwinkel N., Khokhar A. **Multiple Sequence Alignment System for Pyrosequencing Reads**. *Lecture Notes in Computer Science*, **5462/2009**, pp 362-375 (2009). Available http://www.springerlink.com/content/x14j0w4l20268010/ 2, 4, 45, 47, 48, 49, 50, 51, 113, 115, 117, 120, 121, 136, 137, 142

[5] M. Ronaghi. **Pyrosequencing Sheds Light on DNA Sequencing**. *Genome Res.*, **11** pp 3-11 (2001). Available http://genome.cshlp.org/content/11/1/3.full 8, 10

[6] Margulies M. et. al. **Genome Sequencing in Open Microfabricated High Density Picoliter Reactors**. *Nature*, **437** (7057) pp 376-80, (2005). Available http://www.ncbi.nlm.nih.gov/pubmed/16056220 10, 11, 13, 14, 15, 17, 18, 19, 20, 22, 33, 35, 36, 77, 82

[7] Balzer S., Lanzen A., Sharma A., Jonassen I., Malde K. **Characteristics of 454 Pyrosequencing Data - Enabling Realistic Simulation with Flowsim**. *Journal of Bioinformatics*, **26**(18) pp i420-i425, (2010). Available http://bioinformatics.oxfordjournals.org/content/26/18/i420.short 4, 5, 13, 15, 16, 17, 27, 28, 30, 33, 36, 39, 40, 71, 73, 82, 88, 132, 171

[8] Balzer S., Jonassen I., Malde K. **Systematic exploration of error sources in pyrosequencing flowgram data**. *Journal of Bioinformatics*, **27**(13) pp i304i309, (2011). http://bioinformatics.oxfordjournals.org/content/27/13/i304.short 33

[9] Perelson A.S, Markowitz M., Leonard J.M., Ho D., Neumann A.U. **HIV-1 Dynamics in Vivo: Virion Clearance Rate, Infected Cell Life-Span, and Viral Generation Time**. *Science*, **271**(5255) pp 1582-1586, (1996). Available http://www.sciencemag.org/content/271/5255/1582.short 24

[10] M. Nowak. **HIV mutation rate**. *Nature*, **347**(6293):52, (1990). Available http://www.ncbi.nlm.nih.gov/pubmed/2215679 24

[11] Najera R., Quinones-Mateu M.E., Munoz-Fernandez M.A., Lopez-Galindez C., Domingo E., Najera I., Holguin A. **Pol gene quasispecies**

of human immunodeficiency virus: mutations associated with drug resistance in virus from patients undergoing no drug therapy. *J Virol*, **69**(1) pp 23-31, (1995). Available http://jvi.asm.org/content/69/1/23 25

[12] Johnson V.A., Clotet B., Gunthard H.F., Kuritzkes D.R., Pillay D., Shapiro J.M., Richman D.D., Brun-Vezinet F. **Update of the Drug Resistance Mutations in HIV-1**. *Top HIV Med*, **16**(5) pp 138-145, (2008). Available https://wwwnew.iasusa.org/pub/topics/2008/issue5/138.pdf 25

[13] Gharizadeh B. Ronaghi M. Shafer R.W. **Characterisation of mutation spectra with ultra-deep pyrosequencing: application to HIV-1 drug resistance**. *Genome Res.*,**17** pp 1195-1201, (2007). Available http://genome.cshlp.org/content/17/8/1195 4, 46, 59, 112, 126, 134, 137, 148

[14] Tsibris A.M.N., Lee W., Paredes R., Arnaout R., Russ C. et al **Detection and quantification of minority HIV-1 env V3 loop sequences by ultra-deep sequencing: Preliminary results**. *Antivir Ther*, **11**: S74 (2006). Available http://www.aegis.com/conferences/hivdrw/2006/66.html 25

[15] Quince C., Curtis T. P., Davenport R. J., Hall N., Head I. M., Read F. L., Sloan W. T. Lanzen A. **Accurate determination of microbial diversity from 454 pyrosequencing data**. *Nature Methods*, **6**(9) pp 639-41, (2009). Available http://www.ncbi.nlm.nih.gov/pubmed/19668203 4, 33, 41, 67, 68, 70, 132, 161, 171

[16] Richter D. C., Auch A. F., Schmid R., Huson D., Ott F. **MetaSimA Sequencing Simulator for Genomics and Metagenomics**. *PLoS ONE*, **3**(10): e3373, (2008). Available http://www.plosone.org/article/info:doi/10.1371/journal.pone.0003373 33, 35, 36, 39, 82

[17] Quinlan A. R., Stewart D. A., Stromberg M. P., Gabor T. M. **Pyrobayes: an improved base caller for SNP discovery in pyrosequences**. *Nature Methods*, **5** pp 179 - 181 (2008). Available http://www.nature.com/nmeth/journal/v5/n2/abs/nmeth.1172.html 4, 33, 34, 36, 38, 82, 132

[18] Vacic V., Jin H., Zhu J., Lonardi S. **A probabilistic method for small rna flowgram matching**. *Pac Symp Biocomput.* pp 75-86, (2007). Available http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3137517/ 42, 53, 55, 56, 118, 119, 132

[19] Beckstette M., Homann R., Giegerich R., Kurtz S. **Fast index based algorithms and software for matching position specific scoring matrices**. *BioMed Central.*, **7**(389) 2006. Available http://www.biomedcentral.com/1471-2105/7/389 58, 119

[20] Smith T. F., Waterman M. S. **Identification of Common Molecular Subsequences**. *J. Mol. Bwl.* ,**147**(1), pp 195-7. (1981). Available http://www.ncbi.nlm.nih.gov/pubmed/7265238 44, 109, 110, 126, 137

[21] Wikipedia (2011) **List of sequencing alignment software**. *http://en.wikipedia.org/wiki/List_of_sequence_alignment_software*, [Accessed 6 May 2011]. 43

[22] Gotoh O. **An improved algorithm for matching biological sequences**. *J. Mol. Bwl.* ,**162** pp 705-708. (1982). Available http://www.virtual.unal.edu.co/cursos/ingenieria/2001832/lecturas/articulos_exp/gotoh1982.pdf 45

[23] Gentleman R. C., Carey V. J., Bates D. M., Bolstad B., Dettling M., Dudoit S., Ellis B., Gautier L., Ge Y., Gentry J., Hornik K., Hothorn

T., Huber W., Iacus S., Irizarry R., Leisch F., Li C., Maechler M., Rossini A.J., Sawitzki G., Smith C., Smyth G., Tierney L., Yang J.Y.H., Zhang J. **Bioconductor: open software development for computational biology and bioinformatics**. *Genome Biology*, **5**(10):R80. (2004). Available http://www.ncbi.nlm.nih.gov/pubmed/15461798 109

[24] Benson D. A., Boguski M. S., Lipman D. J., Ostell J. **Genbank**. *Nucleic Acids Research*, **25**(1), pp. 1-6 (1997). Available http://nar.oxfordjournals.org/content/25/1/1.short 106

[25] Needleman S.B., Wunsch C.D. **A general method applicable to the search for similarities in the amino acid sequence of two proteins**. *Journal of Molecular Biology*, **48**(3), pp. 443453 (1970). Available http://www.sciencedirect.com/science/article/pii/0022283670900574

[26] Thompson J. D., Higgins D. G., Gibson T. J. **CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice**. *Nucleic Acids Research*, **22**(22) pp. 4673-80 (1994). Available http://www.ncbi.nlm.nih.gov/pubmed/7984417 118

[27] Krznaric D., Levcopoulos C. **Fast Algorithms for Complete Linkage Clustering**. *Discrete and computational geometry*, **19**(1) pp. 131-145 (1998). Available http://www.springerlink.com/content/xhyt12vat971uam6/ 67

[28] Ranganathan A. **The Dirichlet Process Mixture (DPM) Model**, [online] Available at: http://biocomp.bioen.uiuc.edu/journal_club_web/dirichlet.pdf [Accessed 03 January 2011]. 2004. 65

[29] Neal R. M. **Markov Chain Sampling Methods for Dirichlet Process Mixture Models**. *Journal of Computational and Graphical Statistics*, **9**(2) pp. 249-265 (2000). Available http://www.jstor.org/pss/1390653 66

Appendix A

PyroSim Mathematica 8.0 code

A.1 PyroSim Preliminaries: generatePopulation Module

The generatePopulation Module is comprised by three helper methods.

(i) **Method 1: The *mutate* method -** This method takes a nucleotide sequence, ***seq***, and a mutation position, ***pos***, as parameters.

The method selects a random nucleotide from the subset of the alphabet of nucleotide bases that excludes the specific nucleotide at position ***pos*** of ***seq***. Subsequently the nucleotide at position ***pos*** is replaced by the new randomly selected nucleotide.

The Mathematica code for this method is shown in Listing L.1.1.

Listing L.1.1: mutate method

```
mutate[seq_, pos_] := Module[{alphabet, char, newChar, seq2},
   (*installable[] initializes the alphabet to the list
       {"A","C","G","T"}*)
   alphabet = installable[];
```

```
   char = StringTake[seq, {pos}];
   newChar = RandomChoice[Select[alphabet, # != char &]];
   seq2 = Characters[seq];
   seq2[[pos]] = newChar;
   Return[StringJoin[seq2]];
];(*End of mutate method*)
```

(ii) **Method 2: The *variant* method -** This method takes in two parameters; a nucleotide sequence, ***seq***, and a distance, ***dist***. The method creates a variant of ***seq*** that has a mutation distance of approximately ***dist*** from ***seq***, by repeatedly calling the ***mutate*** method.

Given an input sequence ***seq*** of length L, the sequence of attainable mutation distances is $\frac{1}{L}, \frac{2}{L}, \frac{3}{L}, \frac{4}{L}, ..., \frac{L}{L}$, whereby a variant of mutation distance $\frac{i}{L}$ from ***seq*** contains ***i*** substitutions with respect to ***seq***.

The variant method approximates ***dist*** by finding the first element, $\frac{i}{L}$, in the sorted sequence of attainable mutation distances from ***seq*** that has the minimal absolute difference from ***dist***. The mathematica code for the variant method is shown in Listing L.1.2.

Listing L.1.2: variant method

```
variant[seq_, dist_] := Module[{distTable, num, approx, len,
mutatePositions},
   len = StringLength[seq];

   (*The absolute differences between dist and i/len (for all integers i
       in [1,len]) are stored in distTable*)
   distTable = Table[Abs[(i/len) - dist], {i, 1, len}];
```

```
    (*The first number i that yields the minimal absolute distance is
        stored in num*)
    num = First[Flatten[Position[distTable, Min[distTable]]]];

    (*The approximation of dist is num/len and we store this in approx*)
    approx = N[num/len];

    (*We desire num mutations in num distinct positions in seq*)
    mutatePositions = RandomSample[Range[1, len], num];

    (*We derive the desired variant*)
    ret = Fold[mutate, seq, mutatePositions];

     (*We return the variant with information of its true distance from
         seq*)
    Return[{StringJoin["Variant difference = ", ToString[approx],
     "(Approximation of ", ToString[N[dist]], ")"], ret}];
];(*End of variant method*)
```

(iii) **Method 3: The *generateVariant* method -** This method takes three parameters: the original sequence (***seq***), the mutation distance of a variant to be generated from the original sequence (***dist***) and the number of copies of the generated variant that the user requires (***numVariants***). Note that the last two parameters make up a variant specification. The ***generateVariant*** method calls the ***variant*** method to generate a variant, ***v***, which has a mutation distance from ***seq*** that approximately equals ***dist***. The method then generates ***numVariant*** copies of ***v***.

The Mathematica code for the ***generateVariant*** method is shown in Listing

L.1.3

Listing L.1.3: generateVariant method

```
generateVariant[seq_, dist_, numVariants_] := Module[{v},
   v = variant[seq, dist];
   Return[Table[v, {numVariants}]]
];(*End of generateVariant method*)
```

Recall the generic invocation of the ***generatePopulation*** module:

$$\textit{generatePopulation}[\textit{originalSequence},\textit{variantList}].$$

The ***generatePopulation*** module produces either a homogeneous or a heterogeneous population of nucleotide sequences (which are variants of an original sequence) by mapping the ***generateVariant*** method onto each variant of the original sequence specified in the ***variantList*** parameter.

A.2 PyroSim Stage 1: fragmentSequence Module

The Mathematica code for the ***fragmentSequence*** module is shown in Listing L.2.1 below.

Listing L.2.1: fragmentSequence module

```
Module[{fragmentLengths, fragmentPositions, fragmentPositions2,
  fragmentPositionsFinal, len},
 len = StringLength[seq];

 (*We derive random fragment lengths from dist*)
 fragmentLengths =
  Table[Abs[
```

```
    Ceiling[RandomReal[dist]]], {StringLength[seq]/Mean[dist]}];

  (*If we draw a fragment length of 0,
  we increment it to 1 so as to avoid trying to cut a fragment of length 0*)
  Map[If[fragmentLengths[[#]] == 0, fragmentLengths[[#]]++] &,
   Range[Length[fragmentLengths]]];

  (*The list of the cumulative sums of the drawn fragment lengths indicates
      the fragment positions*)
  fragmentPositions = FoldList[Plus, 1, fragmentLengths];

  (*We ensure that the lengths of the fragments we are cutting add up to the
      length of seq*)
  fragmentPositions =
   Append[Sort[Select[fragmentPositions, # < len &]], len];
  fragmentPositions2 = Partition[fragmentPositions, 2, 1];
  fragmentPositionsFinal =
   Append[Map[{First[#], Last[#] - 1} &, Most[fragmentPositions2]],
    Last[fragmentPositions2]];

  (*We cut the fragments and return information on the position of origin of
      each fragment within seq*)
  Map[{StringJoin["Position ", ToString[First[#]], " to ",
      ToString[Last[#]], " of ", seq],
     StringTake[seq, {First[#], Last[#]}]} &, fragmentPositionsFinal]

];(*End of fragmentSequence
module*)
```

A.3 PyroSim Stage 2: sequencingReaction Module

The Mathematica code for the ***sequencingReaction*** module is shown in Listing L.3.1. Here an input DNA fragment and number of flow cycles are stored in the variables ***frag*** and ***cycles*** respectively.

Also in Listing L.3.1. the ***pos*** variable keeps track of the current flow cycle and ensures that the nucleotides in the returned progression obey the contiguous cycling of T, A, C, G.

Listing L.3.1: sequencingReaction module

```
sequencingReaction[frag_, cycles_] := Module[{list, pos = 0, i},
  (*This groups the contiguous identical nucleotides in frag*)
  list = Split[Characters[frag]];

  (*We tally each group of contiguous identical nucleotides*)
  list = Flatten[Map[Tally[#] &, list], 1];

  (*This marks the completion of simulation stage 2 sub-stage 1*)
  (*We now iterate through all the flow cycles ensuring that each flow cycle
      is constrained to the ordering T, A, C, G*)
  pos = 0;
  For[i = 1, i <= cycles, i++,
   If[pos + 1 > Length[list], list = Append[list, {"T", 0}];,
    If[First[list[[pos + 1]]] != "T",
     list = Insert[list, {"T", 0}, pos + 1];];(*End If*)];(*End If*)
   If[pos + 2 > Length[list], list = Append[list, {"A", 0}];,
    If[First[list[[pos + 2]]] != "A",
     list = Insert[list, {"A", 0}, pos + 2];];(*End If*)];(*End If*)
   If[pos + 3 > Length[list], list = Append[list, {"C", 0}];,
    If[First[list[[pos + 3]]] != "C",
```

```
    list = Insert[list, {"C", 0}, pos + 3];];(*End If*)];(*End If*)
   If[pos + 4 > Length[list], list = Append[list, {"G", 0}];,
    If[First[list[[pos + 4]]] != "G",
     list = Insert[list, {"G", 0}, pos + 4];];(*End If*)];(*End If*)
   pos += 4;
   ];(*End For*)
  (*This marks the completion of simulation stage 2 sub-stage 2*)

  (*The code segment below returns a progression of homopolymers
whose length is four times the number of user-
  specified flow cycles (because there are 4 distinct bases T-A-C-
  G in each flow cycle).*)
  Take[list, cycles*4]
  ]; (*End of sequencingReaction module*)
```

A.4 PyroSim Stage 3: flowgram Module

The Mathematica code for the ***flowgram*** module is shown in Listing L.4.1. Here the input homopolymer progression, the number of flow cycles, the original user-defined FVDM, the standard deviation increase gradient and the number of interim flow cycles preceding each flow value degradation are stored in the variables ***homopolymerProg***, ***cycles***, ***distModel***, ***sdIncFactor*** and ***sdInterimFlowCycles*** respectively.

Listing L.4.1: flowgram module

```
flowgram[homopolymerProg_, cycles_, distModel_, sdIncFactor_,
  sdInterimFlowCycles_] := Module[{incFunc},

  (*The incFunc method is used to simulate flow value degradation along the
      length of a flow value progression.
```

```
  It modifies a Gaussian distribution in a FVDM by increasing the spread of
      the distribution*)
  incFunc[dist_, x_] :=
   NormalDistribution[Mean[dist],
    StandardDeviation[dist] + (sdIncFactor*x)];

  (*Below we call the incFunc method to apply degradation to the input FVDM so
       as to obtain the compound FVDM*)
  If[finalModel == {},
   finalModel =
     Map[Flatten[Append[{First[distModel]}, #]] &,
      Transpose[
       Outer[incFunc,
        Rest[distModel], (Range[
           Ceiling[Divide[cycles, sdInterimFlowCycles]]] - 1)]]];
   ];(*End If*)

  (*The code segment below returns the progression of flow values derived from
       the progression of homopolymer lengths using the
compound FVDM*)
  Map[{First[Flatten[Take[homopolymerProg, {# + 1}]]],
     RandomReal[
      finalModel[[Quotient[#, sdInterimFlowCycles*4] + 1,
       Last[Flatten[Take[homopolymerProg, {# + 1}]]] + 1]]]} &,
   Range[0, Length[homopolymerProg] - 1]]
  ]; (*End of flowgram module*)
```

A.5 PyroSim Stage 4

A.5.1 Sub-stage 1 helper methods

A.5.1.1 overlapInterval Method

The Mathematica code for this method is shown in Listing L.5.1.

Listing L.5.1: overlapInterval Method

```
overlapInterval[r_, s_] := Module[{},
  (*Select the range of flow values for the distribution of homopolymer length
      r that have a frequency over 0.05 (according to the PDF)*)
  inv1 =
   Select[Range[0.01, 15, 0.01],
    PDF[flowValueDistModel[[r + 1]], #] > 0.05 &];

  (*Select the range of flow values for the distribution of homopolymer length
      s that have a frequency over 0.05 (according to the PDF)*)
  inv2 =
   Select[Range[0.01, 15, 0.01],
    PDF[flowValueDistModel[[s + 1]], #] > 0.05 &];

  (*Return the intersection of inv1 and inv2*)
  IntervalIntersection[Interval[{First[inv1], Last[inv1]}],
   Interval[{First[inv2], Last[inv2]}]]
  ]; (*End of overlapInterval method*)
```

A.5.1.2 showAmbiguousCycles Method

The Mathematica code for this method is shown in Listing L.5.2.

Listing L.5.2: showAmbiguousCycles method

```
showAmbiguousCycles[flowgram_] := Module[{ed, p},
```

```
(*The ed method takes a flow cycle as a parameter and returns "N" if the
    flow cycle is ambiguous,
otherwise it returns the flow cycle unchanged*)
ed[cycle_] := Module[{vals},
  vals = Map[Last[#] &, cycle];
  If[Length[Select[vals, # < Last[inv1] &]] == 4,
   Return[{{"N"}}];
   ];(*End if*)
  Return[cycle];
  ];(*End of ed method*)

(*Partition flowgram into distinct flow cycles*)
p = Partition[flowgram, 4];

(*The ed method is mapped onto each distinct flow cycle in p*)
Flatten[Map[ed[#] &, p], 1]
]; (*End of showAmbiguousCycles method*)
```

A.5.1.3 trimAmbiguousCycles method

The Mathematica code for this method is shown in Listing L.5.3.

Listing L.5.3: trimAmbiguousCycles method

```
trimAmbiguousCycles[flowgram_] := Module[{pos},
  pos = Position[flowgram, {"N"}];
  If[Length[pos] > 0,
   Return[Take[flowgram, {1, First[Flatten[Sort[pos]]] - 1}]];,
   Return[flowgram];
   ];(*End If*)
  ];(*End of trimAmbiguousCycles method*)
```

A.5.2 Sub-stage 2 - qualityTrim Module

The Mathematica code for this module is shown in Listing L.5.4.

Listing L.5.4: qualityTrim module

```
qualityTrim[flowgram_, errThreshold_, lengthThreshold_] :=
  Module[{err, ret = flowgram},

   (*We find the proportion of flow values in the input flowgram that lie in
       inv, the interval of overlap between the negative flow distribution and
       the unit base incorporation distribution (according to the user supplied
        FVDM)*)
   err =
    N[Length[Select[ret, IntervalMemberQ[inv, Last[#]] == True &]]/
      Length[ret]];

   (*While the proportion of flow values in the overlap region exceeds the
       user-
   specified threshold and while the length of the progression of flow values
       is still long enough, we keep trimming the progression*)
   While[(err > errThreshold) && (Length[ret] > lengthThreshold),
    ret = Most[ret];
    err =
     N[Length[Select[ret, IntervalMemberQ[inv, Last[#]] == True &]]/
       Length[ret]];
    ];

   (*If we exited from the loop and the proportion of flow values in
the the overlap region still exceeds the user-specified threshold,
   the length of the progression must have fallen below the value
lengthThreshold.
```

```
  We ensure that this progression is removed from the pool of
progressions by not returning a result*)
  If[err > errThreshold, ret = {};];
  ret
  ];
```

A.6 PyroSim Stage 5: viewFlowgrams Module

The Mathematica code for this method is shown in Listing L.6.1.

Listing L.6.1: viewFlowgrams module

```
viewFlowgrams[flowgramList_] := Module[{drawFlowgram},

 (*The drawFlowgram method takes a single flowgram as a parameter and draws
    the corresponding flowgram graph*)
 drawFlowgram[flowgramData_] := Module[{},
  (*We call the initColors function to set the following colors to encode
     for the indicated bases: Red->T, Yellow->A, Blue->C,
Green->G*)


  fllcolors = initColors[];
  (*We determine the number of flow values in the flowgram,
  which determines the number of bars in the flowgram graph*)
  lgx = Length[flowgramData];
  (*We determine the highest flow value in the flowgram,
  which determines the height of the longest bar in the flowgram graph*)
  lgy = Max[Map[Last[#] &, flowgramData]];
  (*We draw the flowgram graph*)
  Graphics[{Flatten[{Map[{Last[#],
```

```
        Rectangle[{5.0,
          lgy + (First[Flatten[Position[fllcolors, #]]])/5}, {6.5,
          lgy + ((First[Flatten[Position[fllcolors, #]]])/5) +
           0.1}], Black,
        Text[Style[First[#], Bold], {9.0,
          lgy + ((First[Flatten[Position[fllcolors, #]]])/5) +
           0.05}]} &, fllcolors],
      Map[{Last[
          fllcolors[[
           Mod[First[Flatten[Position[flowgramData, #]]],
            4] /. {0 -> 4}]]],
        Line[{{First[Flatten[Position[flowgramData, #]]],
           0}, {First[Flatten[Position[flowgramData, #]]], #[[
            2]]}}]} &, flowgramData]}]}, Axes -> True,
   PlotRange -> {{0, flowcycles}, {0, lgy + 1}}, AspectRatio -> 1,
   ImageSize -> Medium]
  ];(*End of drawFlowgram method*)

(*The drawFlogram method is mapped onto each flowgram in flowgramList*)
Map[drawFlowgram[#] &, flowgramList]
];(*End of viewFlowgrams module*)
```

A.7 PyroSim Stage 6:

A.7.1 pHomopolymerLengthN

Quince et. al. published a list of empirically observed prior homopolymer probabilities (for homopolymer lengths $\boldsymbol{n}$ where $0 \leq n \leq 9$) accessible at the URL:

http://www.nature.com/nmeth/journal/v6/n9/extref/nmeth.1361-S1.pdf

The pHomopolymerLengthN method returns the prior probability of a supplied homopolymer length $\boldsymbol{n}$ according to this list. The pHomopolymerLengthN method is shown in Listing L.7.1.

Listing L.7.1: pHomopolymerLengthN method

```
pHomopolymerLengthN[n_] :=
  First[Take[{0.5038, 0.3555, 0.1064, 0.02150, 0.007783, 0.002706,
     0.0001344, 0.0001305, 0.00001526, 0.000003815}, {n + 1}]];
```

A.7.2 pSgivenN

Listing L.7.2: pSgivenN method

```
pSgivenN[s_, n_, flowValueDistModel_] :=
  PDF[flowValueDistModel[[n + 1]], s];
```

A.7.3 pNgivenS

Listing L.7.3: pNgivenS method

```
pNgivenS[n_, s_, flowValueDistModel_] :=
  Divide[pSgivenN[s, n, flowValueDistModel]*pHomopolymerLengthN[n],
   Total[Map[(pSgivenN[s, #, flowValueDistModel]*
        pHomopolymerLengthN[#]) &, Range[0, 9]]]];
```

A.7.4 baseCall Module

Listing L.7.4 baseCall module

```
baseCall[flowgram_, compoundDistModel_, sdInterimFlowCycles_] :=
   Module[{call, res, maxN, numList},
```

```
(*The call method takes a flow value and a number indicating which FVDM in
    the compound FVDM to use for base calling.
The method then determines the most likely homopolymer length, h, given the
     flow value according to the indicated FVDM.
It returns the underlying sequence of identical nucleotide bases of length
    h.
Each nucleotide base in the returned sequence has an assigned quality
    score*)
call[flowValue_, modelNum_] := Module[{p},
  p =
   Map[pNgivenS[#, Last[flowValue],
      compoundDistModel[[modelNum]]] &, Range[0, 9]];
  maxN = First[Flatten[Position[p, Max[p]]]];

  (*Return derived nucleotide sequence with quality scores for each
      nucleotide in the sequence*)
  {{First[flowValue], maxN - 1},
   Map[-10*Log[10, 1 - Total[Take[p, {#, Length[p]}]]] &,
    Range[2, maxN]]}
  ];(*End of call method*)

(*We determine the FVDM in the compound FVDM to use for each flow value*)
numList =
 Map[Quotient[#, (sdInterimFlowCycles*4)] &,
   Range[0, Length[flowgram] - 1]] + 1;

(*This code segment maps the call method onto all the flow values in the
    input flow value progression*)
res =
 Map[all[First[#], Last[#]] &,
```

```
    Transpose[{flowgram, numList}]];

  (*Return the sequencing read and accompanying list of quality scores*)
  {StringJoin["Quality Score List: ",
    ToString[Flatten[Map[Last[#] &, res]]]],
   StringJoin[
    Flatten[Map[Table[First[#], {Last[#]}] &, Map[First[#] &, res]]]]}
  ];
```

A.8 PyroSim Main Code

The Main PyroSim code segment is shown in Listing L.8.1 below.

Listing L.8.1: PyroSim main code

```
(*The statements below invoke the modules within PyroSim*)

(*Creation of distributions for the nucleotide sequence fragment
lengths*)
If[distribution == "Normal Distribution",
  dist = NormalDistribution[param1, param2];];
If[distribution == "Uniform Distribution",
  dist = UniformDistribution[{param1, param2}];];

(*-------------------------------------------------------------*)
(*Simulation Stage 1:*)

(*Call to fragmentSequence module to
fragment the input population of nucleotide sequences in preparation
for MPP*)
fragmentPool = Flatten[Map[fragmentSequence[#, dist] &,
```

```
population], 1];
(*-------------------------------------------------------*)



(*-------------------------------------------------------*)
(*Simulation Stage 2:*)

(*Call to sequencingReaction module to
derive a progression of homopolymers from each nucleotide sequence
fragment. The progression of homopolymers respects the ordering of
bases in contiguous flow cycles of "TACG"*)
homoProgPool =
  Map[{First[#], sequencingReaction[Last[#], flowcycles]} &,
   fragmentPool];
(*-------------------------------------------------------*)



(*-------------------------------------------------------*)
(*Simulation Stage 3:*)

(* Call to flowgram module to translate each
progression of homopolymers into a flowgram*)
flowgramPool =
  Map[{First[#],
     flowgram[Last[#], flowcycles, flowValueDistModel, deg,
      degFactor]} &, homoProgPool];
(*-------------------------------------------------------*)



(*-------------------------------------------------------*)
```

```
(*Simulation Stage 4*)

(*Here the overlapInterval module is called
to find the overlap region between a negative flow distribution and
a unit base incorporation distribution. The result is stored in the
public variable inv*)
inv = overlapInterval[0, 1];

(*If the user opts to trim the ambiguous flow cycles in all
flowgrams, this is done in two steps:*)
If[trimA,

  (*Step 1 -
  A call to the showAmbiguousCycles method mapped onto each flowgram in
      flowgramPool*)
  flowgramPool =
   Map[{First[#], showAmbiguousCycles[Last[#]]} &, flowgramPool];

  (*Step 2 -
  A call to the trimAmbiguousCycles method mapped onto each flowgram in
      flowgramPool*)
  flowgramPool =
   Map[{First[#], trimAmbiguousCycles[Last[#]]} &, flowgramPool];

];

(*Call to the qualityTrim module to quality trim each flowgram*)
qTflowgramPool =
  Map[If[Length[Last[#]] != 0, {First[#],
      qualityTrim[Last[#], qtErr, qtThreshold]}, #] &, flowgramPool];
```

```
(*Calculate the proportion of flowgrams that have been filtered out
by quality trimming*)
perc = Length[Select[qTflowgramPool,
Length[Last[#]] == 0 &]]/
    Length[qTflowgramPool]*100;

(*-----------------------------------------------------------*)

(*The simulator returns output according to user specifications*)
If[out == "Flow Value Progressions",
    If[printFlowgram,
        (*-----------------------------------------------------------*)
        (*Simulation Stage 5*)
        (*If the user opts to print a flowgram graph for each flowgram,
        we make a call to the viewFlowgrams module*)
        qTflowgramPool = Select[qTflowgramPool, Length[Last[#]] != 0 &];
        v = viewFlowgrams[Map[Last[#] &, qTflowgramPool]];
        v = Transpose[{qTflowgramPool, v}];
        Map[Map[Print[#, "\n"] &, #] &, v];
        (*-----------------------------------------------------------*)
    ];
    (*Return the list of quality trimmed flowgrams*)
    Return[Select[qTflowgramPool, Length[Last[#]] != 0 &]];,
    If[out == "Sequencing Reads",
        (*-----------------------------------------------------------*)
        (*Simulation Stage 6*)
        (*Pyrosequencing reads are obtained performing base calling on the flow
            values. Each read is accompanied by a list of quality scores
           corresponding to each base in the read*)
```

```
        seqReadPool = Map[baseCall[Last[#], finalModel, degFactor] &,
            qTflowgramPool];
        seqReadPool = Map[Flatten[#, 1] &, Transpose[{Map[First[#] &,
            qTflowgramPool], seqReadPool}]];
        Map[If[StringLength[Last[seqReadPool[[#]]]] != 0, AppendTo[fragmentscp,
             fragmentPool[[#]]]] &, Range[Length[seqReadPool]]];
        seqReadPool = Select[seqReadPool, StringLength[Last[#]] != 0 &];
        (*Calculation of the edit distance between each sequencing read and the
             original fragment from which it was derived*)
        eDist = Map[EditDistance[seqReadPool[[#]][[3]],Last[fragmentscp[[#]]]]
            &, Range[Length[seqReadPool]]];
        (*-------------------------------------------------------*)
        If[printFlowgram,
            (*-------------------------------------------------------*)
            (*Simulation Stage 5*)
            (*If the user opts to print a flowgram graph for each flowgram, we
                make a call to the viewFlowgrams module*)
            qTflowgramPool = Select[qTflowgramPool, Length[Last[#]] != 0 &];
            v = viewFlowgrams[Map[Last[#] &, qTflowgramPool]];
            v = Transpose[{seqReadPool, v}];
            Map[Map[Print[#, "\n"] &, #] &, v];
            (*-------------------------------------------------------*)
        ];
    (*Return the list of sequencing reads*)
    Return[seqReadPool];
    ];
];
```

Note: The PyroSim Main code includes outputs to a simulation log upon the completion of each stage of simulation. These are not included in the listings above.

The full PyroSim code is available in the PyroSim Mathematica 8.0 package.

Appendix B

MPPAlign Mathematica 8.0 code

B.1 Smith Waterman Alignment Module

```
swLocalAlign[seq1_, seq2_, match_: 1, mismatch_: - 1/3, gapOpen_:-1,
  gapExtend_: - 1/3, type_: "Affine", num_: "Many"] :=
 Module[{score, scoreTable, m, p, q, dir, startI, startJ, align,
  gap = "-", ret = {}, it, i, suffix, l1, l2, overallScore = 0},

 (*Call to MPPAlign auxiliary module to set iteration and recursion limits*)
 initLimits[];

 Print["Aligning :", seq1];
 Print[ "with : ", seq2];

 (*The score method assigns scores to alignment matches and mismatches*)
 score[i_, j_] :=
  If[StringTake[seq1, {i}] == StringTake[seq2, {j}], match, mismatch];

 p = StringLength[seq1];
 q = StringLength[seq2];
```

```
(*We initialize the first row and the first column of the scoring matrix to
    0*)
m[0, 0] := 0;
m[i_, 0] := 0;
m[0, j_] := 0;

(*The m method implements dynamic programming to build up the scoring matrix
     and the direction matrix*)
m[i_, j_] := m[i, j] = Module[{pos, max},
   If[type == "Affine",
    pos = {m[i - 1, j - 1] + score[i, j],
       Max[Table[
         m[i, j - k] + (gapOpen + (gapExtend*k)), {k, 1, j}]],
       Max[Table[
         m[i - l, j] + (gapOpen + (gapExtend*l)), {l, 1, i}]], 0};,
    If[type == "Simple",
      pos = {m[i - 1, j - 1] + score[i, j], m[i, j - 1] + gapOpen,
         m[i - 1, j] + gapOpen, 0};
      ];
    ];
   max = Max[pos];
   dir[i, j] = First[First[Position[pos, max]]];
   If[max > overallScore,
    overallScore = max;
    ];
   max
   ];

(*The align method constructs a visual alignment using direction matrix*)
```

```
align[i_, 0] :=
 align[i,
   0] = {{StringTake[seq1, {1, i}], StringJoin[Table[gap, {i}]]}};
align[0, j_] :=
 align[j,
   0] = {{StringJoin[Table[gap, {j}]], StringTake[seq2, {1, j}]}};
align[i_, j_] := align[i, j] =
  If[dir[i, j] == 1,
   Append[
    align[i - 1, j - 1], {StringTake[seq1, {i}],
     StringTake[seq2, {j}]}],
   If[dir[i, j] == 2,
    Append[align[i, j - 1], {gap, StringTake[seq2, {j}]}],
    If[dir[i, j] == 3,
     Append[align[i - 1, j], {StringTake[seq1, {i}], gap}],
     {{StringJoin[Table[gap, {j - i}], StringTake[seq1, {1, i}]],
       StringJoin[Table[gap, {i - j}], StringTake[seq2, {1, j}]]}}
     ]
    ]
   ];

(*We call m to begin a local alignment from the end points of seq1 and seq2.
    The scoring matrix is saved in scoreTable*)
scoreTable = Table[m[i, j], {i, 1, p}, {j, 1, q}];
(*We store the list of all alignments with the maximal score*)
it = Position[scoreTable, Max[scoreTable]];

(*For each alignment with the maximal score...*)
For[i = 1, i <= Length[it], i++,
 (*{startI,startJ} stores the starting point of the alignment*)
```

```
{startI, startJ} = it[[i]];
l1 = StringLength[seq1] - startI;
l2 = StringLength[seq2] - startJ;

(*The suffix stores the substrings of the reference sequence and the
    sequencing read that are to the right hand side of the starting point of
     the alignment*)
suffix = {{StringJoin[StringTake[seq1, -l1],
    StringJoin[Table[gap, {l2 - l1}]]],
   StringJoin[StringTake[seq2, -l2],
    StringJoin[Table[gap, {l1 - l2}]]]}};

(*We construct the alignment by calling the align method and subsequently
    append each subsequence in the suffix to its corresponding aligned
    subsequence*)
ret =
 Append[ret,
  Map[StringJoin[#] &,
   Transpose[Flatten[{align[startI, startJ], suffix}, 1]]]];
];(*End For*)

(*We return the alignment score and the alignment*)
Print["Alignment score: ", N[overallScore]];
If[num == "Single",
 ret = First[ret];
 Print["Alignment result: ", ret // TableForm];,
 Print["Alignment result: "];
 Map[Print[# // TableForm] &, ret];
 ];
```

```
 ClearAll[score, scoreTable, m, p, q, dir, startI, startJ, align,
  gap, it, i, suffix, l1, l2, seq1, seq1];

 {StringJoin["Alignment score = ", ToString[N[overallScore]]], ret}
 ](*End of SWlocalAlign module*)
```

B.2 Assymetric Smith Waterman Alignment Module

```
aswLocalAlign[referenceSequence_, {read_, phredScores_}, match_: 1,
 mismatch_: - 3, gapOpen_: - 5, gapExtend_: - 2, num_: "Many"] :=
Module[{score, scoreTable, m, p, q, dir, startI, startJ, align,
  gap = "-", ret = {}, W, it, i, suffix, l1, l2, overallScore = 0},

 (*Call to MPPAlign auxilliary module to set iteration and recursion limits*)
 initLimits[];

 Print["Aligning :", referenceSequence];
 Print[ "with : ", read];

(*The score method assigns scores to alignment matches and mismatches*)
 score[i_, j_] :=
  If[StringTake[referenceSequence, {i}] == StringTake[read, {j}],
   match, mismatch];

 p = StringLength[referenceSequence];
 q = StringLength[read];

 (*We initialise the first row and the first column of the scoring matrix to
    0*)
```

```
m[0, 0] := 0;
m[i_, 0] := 0;
m[0, j_] := 0;

(*The m method implements dynamic programming to build up the scoring matrix
    and the direction matrix*)
m[i_, j_] := m[i, j] = Module[{pos, max},
  W = 1.0 - (10^(-phredScores[[j]]/10));
  pos = {m[i - 1, j - 1] + (score[i, j]*W),
    Max[Table[
      m[i, j - k] + ((gapOpen + (gapExtend*k))*W), {k, 1, j}]],
    Max[Table[m[i - l, j] + (gapOpen + (gapExtend*l)), {l, 1, i}]],
     0};
  max = Max[pos];
  dir[i, j] = First[First[Position[pos, max]]];
  If[max > overallScore,
   overallScore = max;
   ];
  max
  ];

(*The align method constructs a visual alignment using direction matrix*)
align[i_, 0] :=
 align[i,
  0] = {{StringTake[referenceSequence, {1, i}],
   StringJoin[Table[gap, {i}]]}};
align[0, j_] :=
 align[j,
  0] = {{StringJoin[Table[gap, {j}]], StringTake[read, {1, j}]}};
align[i_, j_] := align[i, j] =
```

```
  If[dir[i, j] == 1,
   Append[
    align[i - 1, j - 1], {StringTake[referenceSequence, {i}],
     StringTake[read, {j}]}],
   If[dir[i, j] == 2,
    Append[align[i, j - 1], {gap, StringTake[read, {j}]}],
    If[dir[i, j] == 3,
     Append[
      align[i - 1, j], {StringTake[referenceSequence, {i}], gap}],
     {{StringTake[referenceSequence, {1, i}],
       StringJoin[Table[gap, {i - j}], StringTake[read, {1, j}]]}}
     ]
    ]
   ];


(*We call m to begin a local alignment from the end points of the reference
    sequence and the sequencing read.
The scoring matrix is saved in scoreTable*)
scoreTable = Table[m[i, j], {i, 1, p}, {j, 1, q}];
(*We store the list of all alignments with the maximal score*)
it = Position[scoreTable, Max[scoreTable]];
(*For each alignment with the maximal score...*)
For[i = 1, i <= Length[it], i++,
 (*{startI,startJ} stores the starting point of the alignment*)
 {startI, startJ} = it[[i]];
 l1 = StringLength[referenceSequence] - startI;
 l2 = StringLength[read] - startJ;
 (*The suffix stores the substrings of the reference sequence and the
     sequencing read that are to the right hand side of the starting point of
      the alignment*)
```

```
 suffix = {{StringJoin[StringTake[referenceSequence, -l1],
     StringJoin[Table[gap, {l2 - l1}]]],
    StringJoin[StringTake[read, -l2],
     StringJoin[Table[gap, {l1 - l2}]]]}};
 (*We construct the alignment by calling the align method and subsequently
     append each subsequence in the suffix to its corresponding aligned
     subsequence*)
 ret =
  Append[ret,
   Map[StringJoin[#] &,
    Transpose[Flatten[{align[startI, startJ], suffix}, 1]]]];
 ];(*End For*)



Print["Alignment score: ", N[overallScore]];
If[num == "Single",
 ret = First[ret];
 Print["Alignment result: ", ret // TableForm];,
 Print["Alignment result: "];
 Map[Print[# // TableForm] &, ret];
 ];


ClearAll[score, scoreTable, m, p, q, dir, startI, startJ, align,
 gap, W, it, i, suffix, l1, l2, referenceSequence, read];


(*We return the alignment score and the alignment*)
{StringJoin["Alignment score = ", ToString[N[overallScore]]], ret}
](*End of ASWlocalAlign module*)
```

B.3 Gap Propagation Module

We implement Gap Propagation as a function (***gapPropagation***) that encapsulates the two stages of the algorithm.

An MPPAlign user can however run each of these stages directly by calling the ***gapPropagationStage1*** function (to implement stage 1 of the algorithm) and the ***gapPropagationStage2*** function (to implement stage 2 of the algorithm).

B.3.1 Gap Propagation Main Function

The main function for the Gap Propagation algorithm is shown below.

```
gapPropagation[referenceSequence_, readList_,
  pairAlignType_: "SG", {pWmatch_: 1, pWmismatch_: - 1/3,
   pWgapOpen_: - 1, pWgapExtend_: - 1/3, pWtype_: "Affine"}] :=
 Module[{result1, alRefs, alReads, result2},

  (*Call to gapPropagationStage1 function*)
  result1 =
   gapPropagationStage1[referenceSequence, readList,
    pairAlignType, {pWmatch, pWmismatch, pWgapOpen, pWgapExtend,
     pWtype}];
  alRefs = Map[#[[2]] &, result1];
  alReads = Map[#[[3]] &, result1];

  (*Call to gapPropagationStage2 function*)
  result2 = gapPropagationStage2[referenceSequence, alRefs, alReads];

  Return[result2];
  ]
```

B.3.2 Gap Propagation Stage 1

The function that implements stage 1 of Gap Propagation is shown below.

```
gapPropagationStage1[referenceSequence_, readList_,
   pairAlignType_: "SG", {pWmatch_: 1, pWmismatch_: - 1/3,
    pWgapOpen_: - 1, pWgapExtend_: - 1/3, pWtype_: "Affine"}] :=
  Module[{al, alignedRefs, alignedReads},

   (*Call to MPPAlign auxilliary module to set iteration and recursion
       limits*)
   initLimits[];

   (*We map the pairwise alignment strategy chosen for stage 1 onto each
       sequencing read in readList*)
   Print[
    "Pairwise aligning each sequencing read with respect to the reference
        genome"];

   (*If the selected pairwise alignment strategy is Smith Waterman alignment
       (SW)*)
   If[pairAlignType == "SW",
    al =
     Map[Flatten[
        swLocalAlign[referenceSequence, #, pWmatch, pWmismatch,
         pWgapOpen, pWgapExtend, pWtype, "Single"]] &, readList];,

   (*Else If the selected pairwise alignment strategy is Semi Global
       alignment (SG)*)
   If[pairAlignType == "SG",
     al =
      Map[Flatten[
```

```
        semiGlobalAlign[referenceSequence, #, pWmatch, pWmismatch,
         pWgapOpen, pWgapExtend, pWtype, "Single"]] &, readList];
    ];
  ];

  ClearAll[referenceSequence, readList, alignedRefs, alignedReads];
  Return[al];
  ];
```

B.3.3 Gap Propagation Stage 2

The function that implements stage 2 of Gap Propagation is shown below.

```
gapPropagationStage2[referenceSequence_, Refs_, Reads_] :=
 Module[{alignedRefs = Refs, alignedReads = Reads, i, j, k,
   gapsPos = {}, t, posF = {}, posBool},

  (*Below we obtain and sort the positions of the gaps in all the aligned
     reference sequences*)
  For[i = 1, i <= Length[alignedRefs], i++,
   gapsPos = StringPosition[alignedRefs[[i]], "-"];
   If[Length[gapsPos] > 0,
    posF = Append[posF, Map[{i, First[#]} &, gapsPos]];
   ];(*End If*)
  ];(*End For*)

  posF = Sort[Flatten[posF, 1], Last[#1] < Last[#2] &];

  (*posBool is a table of booleans indicating whether the gaps at all the
     distinct positions have been propagated.
  It is used to avoid multiple propagation for two or more gaps (from
```

```
    distinct reference sequences) in the same position *)
posBool = Table[False, {StringLength[referenceSequence]*2}];

(*Gap propagation occurs below*)
Print["Propagating gaps through aligned sequencing reads"];
For[i = 1, i <= Length[posF], i++,
 (*If none of the gaps at the gap position has been propagated...*)


 If[! posBool[[Last[posF[[i]]]]],
  (*We iterate through the aligned sequencing reads propagating the gap
     through each read...*)
  For[j = 1, j <= Length[alignedReads], j++,
   If[Last[posF[[i]]] > StringLength[alignedRefs[[j]]],
    alignedRefs[[j]] = StringJoin[alignedRefs[[j]], "-"];
    alignedReads[[j]] = StringJoin[alignedReads[[j]], "-"];,
    If[StringTake[alignedRefs[[j]], {Last[posF[[i]]]}] != "-",
     alignedRefs[[j]] =
      StringInsert[alignedRefs[[j]], "-", Last[posF[[i]]]];
     alignedReads[[j]] =
      StringInsert[alignedReads[[j]], "-", Last[posF[[i]]]];
     (*When we insert a gap into a sequencing read and its corresponding
        reference sequence, we update the information on the position of
        the gaps in the reference sequence*)
     For[k = i + 1, k <= Length[posF], k++,
      If[First[posF[[k]]] == j,
       posF[[k]] = {First[posF[[k]]], Last[posF[[k]]] + 1};
       ];(*End If*)
      ];(*End For*)
     (*After we update the gap position information, we sort the remainder
```

```
        of posF so that we propagate gaps in sequential order*)
     posF =
      Flatten[{Take[posF, {1, i}],
        Sort[Take[posF, {i + 1, Length[posF]}],
         Last[#1] < Last[#2] &]}, 1];
     ];(*End If*)
    ];(*End If*)
   ];(*End For*)

   (*We indicate that we have propagated the gap at this position to avoid
      redundant propagation at a gap position*)
   posBool[[Last[posF[[i]]]]] = True;
   ];(*End If*)
  ];(*End For*)

ClearAll[referenceSequence, Refs, Reads];

(*We return the resulting aligned reference sequences and aligned reads*)
Return[Transpose[{alignedRefs, alignedReads}]];
];
```

B.4 PyroAlign Module

We implement PyroAlign as a function (***pyroAlign***) that encapsulates the four stages of the algorithm.

An MPPAlign user can however run each of these stages directly by invoking:

- The ***pyroAlignStage1*** function (to run stage 1 of the algorithm)
- The ***pyroAlignStage2*** function (to run stage 2 of the algorithm)

- The ***pyroAlignStage3*** function (to run stage 3 of the algorithm)
- ***pyroAlignStage4*** function (to run stage 4 of the algorithm)

B.4.1 PyroAlign Main Function

The main function for the PyroAlign algorithm is shown below:

```
pyroAlign[referenceSequence_, readList_,
  pairAlignType_: "SG", {pWmatch_: 1, pWmismatch_: - 3,
   pWgapOpen_: - 3, pWgapExtend_: - 2,
   pWtype_: "Affine"}, {SWmatch_: 3, SWmismatch_: - 3,
   SWgapOpen_: - 3, SWgapExtend_: - 2,
   SWtype_: "Affine"}, {x_: 5, {PPdnaMatrix_: "CLUSTALW",
    PPgapOpen_: - 50,
    PPgapExtend_: - 5}, {PPdnaMatrixFinal_: "CLUSTALW",
    PPgapOpenFinal_: - 9999, PPgapExtendFinal_: - 9999}},
  profilePairlwiseAlLog_] :=
 Module[{result1, result2, result3, result4},

  (*Call to pyroAlignStage1 function*)
  result1 =
   pyroAlignStage1[referenceSequence, readList,
    pairAlignType, {pWmatch, pWmismatch, pWgapOpen, pWgapExtend,
     pWtype}];

  (*Call to pyroAlignStage2 function*)
  result2 =
   pyroAlignStage2[referenceSequence,
    Map[{#[[2]], #[[3]]} &, Partition[result1, 3]]];

  (*Call to pyroAlignStage3 function*)
```

```
 result3 =
  pyroAlignStage3[referenceSequence,
   Map[Last[#] &, result2], {SWmatch, SWmismatch, SWgapOpen,
    SWgapExtend, SWtype}];

 (*Call to pyroAlignStage4 function*)
 result4 =
  pyroAlignStage4[referenceSequence,
   Map[Last[#] &,
    result3], {x, {PPdnaMatrix, PPgapOpen,
     PPgapExtend}, {PPdnaMatrixFinal, PPgapOpenFinal,
     PPgapExtendFinal}}, profilePairlwiseAlLog];

 ClearAll[result1, result2, result3];
 Return[result4];
]
```

B.4.2 PyroAlign Stage 1

The function that implements stage 1 of PyroAlign is shown below.

```
pyroAlignStage1[referenceSequence_, readList_,
  pairAlignType_: "SG", {pWmatch_: 1, pWmismatch_: - 3,
   pWgapOpen_: - 3, pWgapExtend_: - 2, pWtype_: "Affine"}] :=
 Module[{al},

  (*Call to MPPAlign auxilliary module to set iteration and recursion
      limits*)
  initLimits[];

  (*We map the pairwise alignment strategy chosen for stage 1 onto each
```

```
    sequencing read in readList*)
Print[
 "Pairwise aligning each sequencing read with respect to the reference
    genome"];

(*If the selected pairwise alignment strategy is Smith Waterman alignment
    (SW)*)
If[pairAlignType == "SW",
 al =
  Flatten[Map[
   swLocalAlign[referenceSequence, #, pWmatch, pWmismatch,
    pWgapOpen, pWgapExtend, pWtype, "Single"] &, readList]];,

 (*Else If the selected pairwise alignment strategy is Semi Global
    alignment (SG)*)
 If[pairAlignType == "SG",
  al =
   Flatten[
    Map[semiGlobalAlign[referenceSequence, #, pWmatch,
      pWmismatch, pWgapOpen, pWgapExtend, pWtype, "Single"] &,
     readList]];
  ];
 ];

ClearAll[referenceSequence, readList];
Return[al];
];
```

B.4.3 PyroAlign Stage 2

The function that implements stage 2 of PyroAlign is shown below.

```
pyroAlignStage2[referenceSequence_, overlappedReads_] :=
  Module[{reorderFunction, reorderedReads},

   (*The reorder function method sorts the aligned reads according to their
       starting position with respect to the reference sequence*)
   reorderFunction[in_] := Module[{out},
     out =
      Table[{First[in[[i]]], contentPos[Last[in[[i]]]],
        Last[in[[i]]]}, {i, 1, Length[in]}];
     out =
      Flatten[Table[
        Select[out, First[#[[2]]] == i &], {i, 1,
         StringLength[referenceSequence] + 25}], 1]
     ];

   (*We reorder the input list of reads*)
   reorderedReads =
    Map[{First[#], Last[#]} &, reorderFunction[overlappedReads]];

   ClearAll[referenceSequence, overlappedReads];
   Return[reorderedReads];
   ];
```

B.4.4 PyroAlign Stage 3

The function that implements stage 3 of PyroAlign is shown below.

```
pyroAlignStage3[referenceSequence_,
```

```
  reorderedReads_, {SWmatch_: 3, SWmismatch_: - 3, SWgapOpen_: - 3,
   SWgapExtend_: - 2, SWtype_: "Affine"}] :=
 Module[{part, pairwiseAligned = {}, SWal, rest = {}, finalPart},

  (*We partition the list of reordered reads into a list of pairs*)
  part = Partition[reorderedReads, 2];
  If[! MemberQ[Flatten[part], Last[reorderedReads]],
   part = Append[part, {Last[reorderedReads]}]];

  (*We pairwise align each pair of neightbouring aligned reads using Smith
      Waterman alignment*)
  SWal =
   Map[If[Length[#] > 1,
      swLocalAlign[First[#], Last[#], SWmatch, SWmismatch, SWgapOpen,
        SWgapExtend, SWtype, "Single"], #] &, part];

  ClearAll[referenceSequence, reorderedReads, part];
  Return[SWal];
  ];
```

B.4.5 PyroAlign Stage 4

pyroAlignStage4, the function that implements stage 4 of PyroAlign, functions with the help of ***profilePairwiseAlign***, a private function internal to MPPAlign.

Subsections B.4.5.1 and B.4.5.2 provide listing of the Mathematica code for ***pyroAlignStage4*** and ***profilePairwiseAlign*** respectively.

B.4.5.1 *pyroAlignStage4* Mathematica code

```
pyroAlignStage4[referenceSequence_,
```

```
 SWal_, {x_: 5, {PPdnaMatrix_: "CLUSTALW", PPgapOpen_: - 50,
   PPgapExtend_: - 5}, {PPdnaMatrixFinal_: "CLUSTALW",
   PPgapOpenFinal_: - 9999, PPgapExtendFinal_: - 9999}},
 profilePairlwiseAlLog_] :=
Module[{part, rest = {}, finalPart, swalcp = SWal, result},

 (*We divide the list of pairwise aligned sequencing reads in to x
    partitions*)
 part = Partition[swalcp, x];

 (*These statements ensure that all we do not loose any sequence read data
    during the partitioning*)
 Map[If[! MemberQ[Flatten[part, 1], #], rest = Append[rest, #]] &,
  swalcp];
 part = Select[Append[part, rest], Length[#] >= 1 &];

 (*We open a log file for the profile-profile pairwise alignments*)
 fileName = profilePairlwiseAlLog;
 f = JavaNew["java.io.File", JavaNew["java.lang.String", fileName]];
 fstream = JavaNew["java.io.FileWriter", f];
 outt = JavaNew["java.io.BufferedWriter", fstream];
 str = "";

 (*Note that for the two statements below, profile-profile pairwise alignment
     is conducted by calling the profilePairwiseAlign
 function, which is a private function within MPPAlign*)

 (*We conduct profile-
 profile alignment within each partition stored in the variable named part.
    This results in each partition being resolved into a profile i.e. a
```

```
    multiple alignment of sequencing reads)*)
 finalPart =
  Table[Fold[
    profilePairwiseAlign[#1, #2, PPdnaMatrix, PPgapOpen,
      PPgapExtend] &, First[part[[i]]], Rest[part[[i]]]], {i, 1,
    Length[part]}];

 (*We conduct profile-
 profile alignment across the profiles obtained in the previous step*)



 result =
  Fold[profilePairwiseAlign[#1, #2, PPdnaMatrixFinal, PPgapOpenFinal,
      PPgapExtendFinal] &, First[finalPart], Rest[finalPart]];

 (*We write to the profile-profile alignment log*)
 outt@write[JavaNew["java.lang.String", ToString[str]]];
 outt@close[];

 ClearAll[referenceSequence, swalcp, part, finalPart];
 Return[result];
 ]
```

B.4.5.2 *profilePairwiseAlign* Mathematica code

```
profilePairwiseAlign[p1_, p2_, dnaMatrix_: "CLUSTALW", gapOpen_: -
50,
   gapExtend_: - 5] :=
 Module[{i, temp1, temp2, temp3, out1, out2, cmd, result},
  initLimits[];
```

```
  LoadJavaClass["java.io.File"];
  LoadJavaClass["java.lang.System"];
  Print["Aligning"];
  str = StringJoin[str,
    "\n\n\
**********************************************************************\
\n\tALIGNING:\n"];
  Print[p1 // TableForm];
  For[i = 1, i <= Length[p1], i++,
   str = StringJoin[str, "\t\t", p1[[i]], "\n"];
   ];
  Print["\t with"];
  Print[p2 // TableForm];
  str = StringJoin[str, "\t WITH:\n"];
  For[i = 1, i <= Length[p2], i++,
   str = StringJoin[str, "\t\t", p2[[i]], "\n"];
   ];

  (*The statements below put profile p1 into a Fasta file to be used as input
     for clustal profile pairwise alignment*)
  temp1 = File`createTempFile["profile1", "fasta"];
  out1 =
   JavaNew["java.io.BufferedWriter",
    JavaNew["java.io.FileWriter", temp1]];
  out1@write[
    StringJoin[
     Map[StringJoin[">", StringJoin["p1_", ToString[#]],
        System`getProperty["line.separator"], p1[[#]],
        System`getProperty["line.separator"]] &, Range[Length[p1]]]]];
  out1@close[];
```

```
Print["Profile 1 saved in ", temp1@toString[]];

(*The statements below put profile p2 into a fasta file to be used as input
    for clustal profile pairwise alignment*)
temp2 = File`createTempFile["profile2", "fasta"];
out2 =
 JavaNew["java.io.BufferedWriter",
  JavaNew["java.io.FileWriter", temp2]];
out2@write[
  StringJoin[
   Map[StringJoin[">", StringJoin["p2_", ToString[#]],
      System`getProperty["line.separator"], p2[[#]],
      System`getProperty["line.separator"]] &, Range[Length[p2]]]]];
out2@close[];
Print["Profile 2 saved in ", temp2@toString[]];

(*temp3 is the clustal pairwise profile alignment output file*)
temp3 = File`createTempFile["result", "fasta"];

(*Call to clustalw2 with the appropriate parameters*)
Print["Calling clustalw2 ..."];
cmd = StringJoin["clustalw2 -PROFILE -TYPE=DNA -DNAMATRIX=",
  dnaMatrix, " -GAPOPEN=", ToString[-gapOpen], " -GAPEXT=",
  ToString[-gapExtend], " -output=fasta -profile1=",
  temp1@toString[], " -profile2=", temp2@toString[], " -outfile=",
  temp3@toString[]];
Print["clustalw2 command is: ", cmd];
Run[cmd];
Print["Clustal result saved in:", temp3@toString[]];
```

```
 (*Print and return the result as a list of sequences*)
 result = Last[importFasta[temp3@toString[]]];
 str = StringJoin[str, "\t RESULT IS:\n"];
 Print["Result is ", result // TableForm];
 For[i = 1, i <= Length[result], i++,
  str = StringJoin[str, "\t\t", result[[i]], "\n"];
  ];
 result
 ]
```

B.5 Flowgram Matching Module

```
flowgramMatching[flowgramList_, flowValueStream_, FVDM_,
param4Type_,
   param4_] :=
  Module[{longestCommonPrefix, skip, skipChain, M,
    computeThresholdScore, suffixes, lexicographicalOrdering, suf,
    lcp, skp, match, result},
   initLimits[];

   (*Note that the input variables are supposed to store the following*)
   (*1. flowgramList - A list of input flowgrams,
   whereby we wish to align each flowgram with respect to flowValueStream*)
   (*2. flowValueStream -
   A flow space representation of a reference genome sequence*)
   (*3. FVDM - A flow value distribution model*)
   (*4. param4Type -
   A parameter that can be set to "SCORE" or "P-VALUE".
   param4Type describes the value stored in param4*)
```

```
(*5. param4 -
A parameter that stores a threshold value that helps to identify
    significant alignments of a flowgram with respect to flowValueStream*)



(*-------------------------------------------------------*)
(*Methods used to compute Enhanced Suffix Array tables*)
(*-------------------------------------------------------*)
(*1. longestCommonPrefix method used to compute the lcp table*)
longestCommonPrefix[list1_, list2_] := Module[{i, ret},
  ret = 0;
  For[i = 1, i < Length[list1], i++,
   If[list1[[i]] != list2[[i]],
     ret = i - 1;
     Break[];
     ];
   ];
  ret
  ];(*End of longestCommonPrefix method*)

(*2. skip method used to compute the skp table*)
skip[list_, i_] := Module[{j, ret},
  ret = Length[list];
  For[j = i + 1, j <= Length[list], j++,
   If[list[[j]] < list[[i]],
     ret = Min[j, Length[list]];
     Break[];
     ];
   ];
  ret
```

```
  ];(*End of skip method*)
(*-------------------------------------------------------*)


(*-------------------------------------------------------*)
(*The skipChain method is used to skip certain ranges of suffixes in the
    table suf*)
skipChain[lcp_, skp_, n_, i_, d_] := Module[{j},
  If[i < n,
   j = i + 1;
   While[j < n && lcp[[j]] > d,
    j = skp[[j]];
    ];,
   j = n + 1;
   ];
  j
  ];(*End of skipChain method*)
(*-------------------------------------------------------*)


(*-------------------------------------------------------*)
(*The M method calculates the score of matching the flow value at position
    i in the input flowgram to the homopolymer length at position j in the
    input flow value stream*)
M[i_, j_] := Module[{mm, p, q},
  mm[x_, y_] :=
   Log[pSgivenN[x, y, FVDM]] -
    Log[Total[
      Map[(pSgivenN[x, #, FVDM]*pHomopolymerLengthN[#]) &,
       Range[0, 9]]]];
  (*A score of -9999 is used to identify a flow value that is being matched
      with a homopolymer that corresponds to a different nucleotide type*)
```

```
  If[First[i] != StringTake[j, 1], Return[-9999]];
  p = Last[i];
  q = ToExpression[StringTake[j, -1]];
  mm[p, q]
  ];(*End of M method*)
(*-------------------------------------------------------*)


(*-------------------------------------------------------*)
(*The computeThresholdScore method is used to convert the input p-
value into a list of threshold scores*)
computeThresholdScore[flowgram_, type_, param_] :=
 Module[{u, PSSM, ss, scMin, scMax, list, continue = True, di = 0,
   p, cumulativeP = 0, th, maxd, thetad, ret},
  u = Union[Most[flowValueStream]];
  PSSM = Round[Outer[M, flowgram, u, 1]];
  maxd = Map[Max[#] &, PSSM];
  thetad =
   Table[Total[Take[maxd, {x + 1, Length[maxd]}]], {x, 1,
     Length[maxd]}];
  If[type == "SCORE",
   Return[param - thetad];
   ];
  scMax = Total[maxd];
  scMin =
   Total[Map[Min[#] &, Map[Select[#, # != -9999 &] &, PSSM]]];
  p[0, t_] := If[t == 0, 1, 0];
  p[i_, t_] := p[i, t] = Module[{r, val, k},
    r = 0;
    For[k = 1, k <= Length[PSSM[[i]]], k++,
     val = PSSM[[i, k]];
```

```
      If[val != -9999,

       r = r + (p[i - 1, t - val]*
            pHomopolymerLengthN[
             ToExpression[StringTake[u[[k]], -1]]]);
       ];
      ];
     r
     ];
  While[continue,
   ss = p[Length[PSSM], scMax - di];
   cumulativeP += ss;
   If[cumulativeP > param,
    th = scMax - di + 1;
    Break[];,
    di++;
   ];
  ];
  ret = th - thetad;
  Print["Threshold score corresponding to p-value ", param, " is ",
    Last[ret]];
  Return[ret];
  ];(*End of computeThresholdScore method*)
(*-------------------------------------------------------*)

(*-------------------------------------------------------*)
(*The match method attempts to match a flowgram to the different suffixes
    of the flow space representation of the reference genome*)
match[fl_] :=
 Module[{flowgram, v, l, n, d, m, C, it, depth, score,
```

```
 suffixLength, value, matchPos = {}, thr},
Print["-----------------------------------------------"];
Print["-----------------------------------------------"];
Print["Beginning to attempt matches for flowgram ", fl];

flowgram = flowgramList[[fl]];
n = Length[flowValueStream];
m = Length[flowgram];
C = Table[0, {m}];

(*We call the computeThresholdScore method to convert the inputted p-value
    into a threshold score*)
If[param4Type == "SCORE",
 Print["Threshold score for alignment of flowgram ", fl, " is ",
   param4];,
 Print["Converting input p-value into list of threshold scores for
    flowgram ", fl, "..."];
 ];
thr = computeThresholdScore[flowgram, param4Type, param4];
Print[thr];
(*The code below adapted from the Beckstette et. al. paper*)
Print["Iterating through lexicographical ordering of suffixes"];
it = 1;
depth = 0;
While[it < n,
 Print["-------------------------------------------"];
 Print["Attempting to match flowgram ", fl, " with suffix number ", it];
 suffixLength = n - suf[[it]] + 1;

 While[(suffixLength < m ||
```

```
    StringTake[lexicographicalOrdering[[it, 1]], 1] !=
     First[Take[flowgram[[1]], 1]]),
   If[suffixLength < m,
    Print[
     "Skipping suffix - Suffix is shorter than input flowgram"];,
    Print[
     "Skipping suffix - Suffix begins with a nucleotide different from the
          input flowgram"];
    ];
   it++;
   If[it >= n,
    matchPos = Sort[matchPos, Last[#1] > Last[#2] &];
    Print["Sorted list of match positions of the input flowgram w.r.t the
        flow-space representation of the reference sequence (Note: Each
        entry in the sorted list is in the format {match position, match
        score}):"];
    Print[matchPos];
    Return[matchPos];,
    Print["-------------------------------------------"];
    Print["Attempting to match flowgram ", fl,
     " with suffix number ", it];
    ];
   depth = Min[depth, lcp[[it]]];
   suffixLength = n - suf[[it]] + 1;
   ];

  If[depth == 0,
   score = 0;,
   score = C[[depth]];
   ];
```

```
d = depth + 1;
While[(d <= m),
 v = flowgram[[d]];
 l = lexicographicalOrdering[[it, d]];
 If[l == "$", Break[];,
  score = score + Round[M[v, l]];
  ];
 C[[d]] = score;
 If[score >= thr[[d]],
  If[d == m,
    Break[];,
    d++;
    ];,
  d--;
  Break[];
  ];
 ];
If[(d == m) && (score >= thr[[m]]),
 matchPos = Append[matchPos, {suf[[it]], score}];
 Print["Match found at position: ", suf[[it]], " Score is: ",
  score];
 While[it <= n,
  it++;
  If[lcp[[it]] >= m,
   matchPos = Append[matchPos, {suf[[it]], score}];
   Print["Match found at position: ", suf[[it]], " Score is: ",
     score];,
   Break[];
   ];
  ];,
```

```
    Print["Match score is below threshold"];
    it = skipChain[lcp, skp, n, it, d];
    ];
   depth = lcp[[it]];
   ];

  matchPos = Sort[matchPos, Last[#1] > Last[#2] &];
  Print["Sorted list of match positions (Note: Each entry in the sorted list
       is in the format {match position, match score}):"];
  Print[matchPos];
  Return[matchPos];
  ];(*End of match method*)



(*-------------------------------------------------------*)
(*-------------------------------------------------------*)
(*flowgramMatching Main Code*)
(*-------------------------------------------------------*)

(*We list all the suffixes in the input flow-
space representation of the reference genome*)
suffixes =
 Map[Join[Take[flowValueStream, -#],
    Table["$", {Length[flowValueStream] - #}]] &,
  Range[Length[flowValueStream]]];
lexicographicalOrdering = Sort[suffixes];
Print["Suffixes of the flow space representation of the reference genome
    generated."];
Print["The suffixes in lexicographical ordering are: ",
 lexicographicalOrdering // TableForm];
```

```
(*We compute the Enhanced Suffix Array tables*)
Print["Generating Enhanced Suffix Array tables."];
suf = Map[Count[#, "$"] &, lexicographicalOrdering];
Print["suf table: ", suf];
lcp =
 Map[longestCommonPrefix[lexicographicalOrdering[[# - 1]],
    lexicographicalOrdering[[#]]] &,
  Range[Length[flowValueStream]]];
Print["lcp table: ", lcp];
skp = Map[skip[lcp, #] &, Range[Length[flowValueStream]]];
Print["skp table: ", skp];

(*We map the match method onto each flowgram in flowgramList and save the
    result*)
result = Map[match[#] &, Range[Length[flowgramList]]];

ClearAll[skp, lcp, suf, lexicographicalOrdering, suffixes];
Return[result];
];
```

Appendix C

MPPErrorCorrect Mathematica 8.0 code

C.1 wangEC Module

```
wangEC[orig_, indeces_, alignedReads_, m1_: 0.0044, m2_: 0.0007,
  flankingHomopolymerWindow_: 5] :=
 Module[{meanN = m2, meanH = m1, al, all, orig2, indeces2,
   alignedReads1, alignedReads2, alignedReads3, max, ex,
   calculatePValue, columns, r1, r2, r3, i, j, tally, tally2, N,
   pValues, sel, out},

  (*The calculatePValue method is used to determine the significance of a
     mutation at a given position of the input multiple alignment*)
  calculatePValue[region_, n_, N_] := Module[{lambda},

    (*If the region is homopolymeric (H), we compute the expected error rate
       (lambda) using the mean error rate for homopolymeric regions*)
    If[region == "H",
```

```
    lambda = meanH*N;,

    (*Else If the region is nonhomopolymeric (N), we compute the expected
        error rate (lambda) using the mean error rate for nonhomopolymeric
        regions*)
    If[region == "N",
      lambda = meanN*N;
      ];
    ];

   (*Return the p-value that corresponds to the mutation*)
   Return[
    Total[Table[Divide[(E^lambda)*(lambda^k), k!], {k, n, N}]]];
   ];(*End of calculatePValue method*)

al = Table[{indeces[[i]], orig[[i]], alignedReads[[i]],
   alignedReads[[i]]}, {i, 1, Length[alignedReads]}];
all = Select[al, Length[contentPos[Last[#]]] > 1 &];
indeces2 = Map[#[[1]] &, all];
orig2 = Map[#[[2]] &, all];
alignedReads1 = Map[#[[3]] &, all];
alignedReads2 = Map[#[[4]] &, all];

(*Here we distinguish leading and trailing gaps "X" from gaps internal to
    the sequencing reads for each sequencing read*)
alignedReads3 =
 Map[StringJoin[Table["X", {First[contentPos[#]] - 1}],
    StringTake[#, contentPos[#]],
    Table["X", {StringLength[#] - Last[contentPos[#]]}]] &,
  alignedReads2];
```

```
(*Here we ensure that all the inputted sequencing reads are of the same
    length*)
max = Max[Map[StringLength[#] &, alignedReads3]];
ex = Table[
  max - StringLength[alignedReads3[[i]]], {i, 1,
   Length[alignedReads3]}];
alignedReads3 =
 Table[StringJoin[alignedReads3[[i]], Table["X", {ex[[i]]}]], {i,
   1, Length[ex]}];

(*We view the list of sequencing reads by column.
This means that we assume the list of sequencing reads is a multiple
    alignment and we view this alignment one position at a time*)


columns = Transpose[Map[Characters[#] &, alignedReads3]];

(*This sequence of calculations demarcates the homopolymeric and
    nonhomopolymeric regions*)
r1 = Map[If[Length[#] > 0, First[First[#]]] &,
  Map[Sort[Select[Tally[#], First[#] != "-" && First[#] != "X" &],
     Last[#1] > Last[#2] &] &, columns]];
r2 = Map[{First[#], Length[#]} &, Split[r1]];
r3 = Flatten[
  Map[If[Last[#] >= 3, Table["H", {Last[#]}],
     Table["N", {Last[#]}]] &, r2]];
If[flankingHomopolymerWindow > 0,
 For[i = 1, i <= Length[r3] - flankingHomopolymerWindow, i++,
  If[r3[[i + flankingHomopolymerWindow]] == "H",
```

```
    For[j = 0, j < flankingHomopolymerWindow, j++,
      r3[[i + j]] = "H";
      ];
    ];
  ];
 For[i = Length[r3], i >= 2, i--,
  If[r3[[i - flankingHomopolymerWindow]] == "H",
    For[j = 0, j < flankingHomopolymerWindow, j++,
      r3[[i - j]] = "H";
      ];
    ];
  ];
 ];

(*This sequence of calculations determines the significant bases at each
    column of the multiple alignment of sequences*)
tally =
 Map[Select[Tally[#], First[#] != "-" && First[#] != "X" &] &,
  columns];
tally2 = Map[Select[Tally[#], First[#] != "X" &] &, columns];
N = Map[Total[Map[Last[#] &, #]] &, tally2];
pValues =
 Table[Table[{First[tally[[p, q]]],
    calculatePValue[r3[[p]], Last[tally[[p, q]]], N[[p]]]}, {q, 1,
    Length[tally[[p]]]}], {p, 1, Length[tally]}];
out = Map[Select[#, Last[#] < 0.001 &] &, pValues];

(*Our output includes a list of the significant bases at each column and
    the error corrected reads*)
{out, indeces2, orig2, alignedReads1,
```

```
  Map[StringJoin[#] &,
   Transpose[
    Table[Map[
      If[! MemberQ[Map[First[#] &, out[[i]]], #],
       If[((Length[out[[i]]] == 1) && (# != "X") && (# != "-")),
        First[First[out[[i]]], "-"], #] &, columns[[i]]], {i, 1,
      Length[columns]}]]]}
 ];
```

C.2 ErikssonEC Module

```
erikssonEC[alignment_, orig_, indeces_, e_: 0.0025, alpha_: 0.001,
  windowLength_: 7, windowOs_: 7] :=
 Module[{alignedReads, alignedReads2, max, ex, windowLengthA, len,
   cP, windowCorrect, fullData, orig2, fullDataIndeces, columns, tWR,
    tWP, w1p1, w1p2, w2p1, w2p2, w3p1, w3p2, eW = {}, wC = {}, wCT,
   wCT2, columnCoverage, cv, cols, seq, chars, currChar, result},

  (*********************************************************************)
  (*             WINDOWCORRECT MODULE USED BY MAIN CODE                *)
  (*             (Scroll down for ErikssonErrorCorrect main code)      *)
  (*                                                                   *)
  (*The windowCorrect module reads in windowStart, windowEnd and wLength, *)
  (*which are the start position, end position and length of the error   *)
  (*correction window respectivelly.                                    *)
  (*********************************************************************)
  windowCorrect[windowStart_, windowEnd_, wLength_] :=
   Module[{window, windowEnd2, splitWindow, splitPos, windowSeq,
     window2, var, consensus, d, sig, mutations, numVariants = 1,
```

```
 BinomPGEQ, NumCoocurencesEQ, mutationPair, pVals1,
 alphabet = Characters["ACGT"], mutations2, t1, p, q,
 evaluated = {}, i, j, lists, pos, windowClusters, indexedData,
 cc1, cc2, windowClustersBySeq, clusterList, seqCt, clusterList2,
  tallies, clusterColumnVar, clusterConsensus, currentSeq,
 correctedReads},

(*BinomPGEQ method used by windowCorrect module to determine the
    significance of a variant that occurs x times in a column given that
    the number of reads that overlap in the column is d*)
BinomPGEQ[x_] :=
 Total[Table[
   Binomial[d, k]*Power[e, k]*Power[1 - e, d - k], {k, x, d}]];

(*NumCoocurencesEQ method used by windowCorrect module to determine the
    significance of a pair of mutations that co-occur in different
    alignment positions in an aligned sequencing read*)
NumCoocurencesEQ[nu_, nv_, c_] :=
 Divide[Binomial[nv, c]*Binomial[d - nv, nu - c],
  Binomial[d, nu]];

(*mutationPair method used by windowCorrect module*)
mutationPair[seq_, i1_, i2_] :=
 Module[{c1, c2, nu, nv, count = 1, n, mutationsCols},
  c1 = mutations2[[seq, i1]];
  c2 = mutations2[[seq, i2]];
  If[! MemberQ[evaluated, {{i1, c1}, {i2, c2}}],
   nu = Last[Flatten[Select[mutations[[i1]], First[#] == c1 &]]];
   nv = Last[Flatten[Select[mutations[[i2]], First[#] == c2 &]]];
   For[n = seq + 1, n <= Length[mutations2], n++,
```

```
    If[(mutations2[[n, i1]] == c1) && (mutations2[[n, i2]] == c2),
      count++;
      ];
    ];
   AppendTo[evaluated, {{i1, c1}, {i2, c2}}];
   If[NumCoocurencesEQ[nu, nv, count] < alpha,
    numVariants++;
    If[Divide[count, Max[nu, nv]] > 0.85,
     numVariants = numVariants - 2;
     ];
    ];
   ];
  ];


(*********************************************************************)
(*WindowCorrect Module Main Code                                     *)
(*********************************************************************)

If[windowEnd > Length[columns],
 (*If the end position of window is greater than available columns, we
     wrap the window around to the begining columns*)
 windowEnd2 = wLength - (Length[columns] - windowStart + 1);
 window =
  Join[Take[columns, {windowStart, Length[columns]}],
   Take[columns, {1, windowEnd2}]];

 Print["Error correction window = Columns: ", windowStart,
  " to ", Length[columns], " and ", 1, " to ", windowEnd2];
```

```
(*We take note of the positions that have been covered by the window*)
Table[
 If[IntervalMemberQ[Interval[{windowStart, Length[columns]}],
   i], columnCoverage[[i]]++;], {i, 1, Length[columnCoverage]}];
Table[
 If[IntervalMemberQ[Interval[{1, windowEnd2}], i],
  columnCoverage[[i]]++;], {i, 1, Length[columnCoverage]}];
(*The window is a split window because it jointly covers some terminating
    columns and some initiating columns in the column list*)
splitWindow = True;
,

(*Else if the span of the window is within the column range then leave
   the span of the window unaltered*)
window = Take[columns, {windowStart, windowEnd}];

Print["Error correction window = Columns: ", windowStart,
 " to ", windowEnd];

(*We take note of the positions that have been covered by the window*)
Table[
 If[IntervalMemberQ[Interval[{windowStart, windowEnd}], i],
  columnCoverage[[i]]++;], {i, 1, Length[columnCoverage]}];

(*The window is not a split window*)
splitWindow = False;
];(*End If*)

(*Transpose the window so that we view it by sequence*)
windowSeq = Transpose[window];
```

```
(*Select the sequences whose content positions occupy at least 65\% of the
     window (A read is selected if the fraction of X's in the read that
    are within the window is less than 0.35)*)
indexedData =
 Select[Table[windowSeq[[i]] -> i, {i, 1, Length[windowSeq]}],
  Divide[Count[First[#], "X"], wLength] <= (0.35) &];
Print["Sequences selected for clustering: "];
Print[indexedData];
If[Length[indexedData] == 0, Return[{}]];

(*Transpose the selected sequences so that the window is a view of the
    columns of the sequences*)
window2 = Transpose[Map[First[#] &, indexedData]];

(*We note the nucleotide variation in each column*)
var =
 Map[Select[
    Tally[#], ((First[#] != "X") && (First[#] != "-")) &] &,
  window2];

(*d stores the number of reads that we are evaluating in this window*)
d = Max[Map[Total[Map[Last[#] &, #]] &, var]];

(*We determine the consensus sequence of the window*)
consensus =
 Table[sig = Select[var[[q]], BinomPGEQ[Last[#]] < alpha &];
  If[Length[sig] > 0,
   First[First[Sort[sig, Last[#1] > Last[#2] &]]], "-"], {q, 1,
   Length[var]}];
```

```
Print["Window consensus ", consensus];

(*We determine the mutations at each nucleotide position in the window*)
mutations =
 Table[Select[var[[i]], First[#] != consensus[[i]] &], {i, 1,
   Length[window2]}];

(*********************************************************************)
(*Statistical test 1:                                                *)
(*   Counting the significant mutations in each column of the window *)
(*********************************************************************)

(*We determine the significance (p-value) of each mutation by calling the
    BinomPGEQ method*)
pVals1 =
 Map[Map[{First[#], BinomPGEQ[Last[#]]} &, #] &, mutations];

(*For loop structure iterates through p-values for list of mutations and
    increases count of number of variants (numVariants) where it finds
    significant mutations (i.e. where a p-value is less than alpha)*)
For[p = 1, p <= Length[pVals1], p++,
 For[q = 1, q <= Length[pVals1[[p]]], q++,
   If[Last[pVals1[[p, q]]] < alpha,
    numVariants++;
    ];
   ];
 ];

(*********************************************************************)
(*Statistical test 2:                                                *)
```

```
(*    Counting the significance of co-occuring mutations within each *)
(*     sequence in the window                                        *)
(*********************************************************************)

(*We transpose columns to view the window as a list of sequences and mark
    the mutations in each sequence (non-mutant bases are marked by "*")*)
mutations2 =
 Transpose[
  Table[Map[If[(# == consensus[[i]]), "*", #] &,
    window2[[i]]], {i, 1, Length[window2]}]];

(*We note the positions of the mutations in each sequence in the window*)
t1 =
 Table[Flatten[
   Position[Map[MemberQ[alphabet, #] &, mutations2[[i]]],
    True]], {i, 1, Length[mutations2]}];

(*This code segment tests each co-occurring mutation pair for significance
     by calling the mutationPair method*)
For[i = 1, i <= Length[t1], i++,
 If[Length[t1[[i]]] >= 2,
   lists = Subsets[t1[[i]], {2}];
   For[j = 1, j <= Length[lists], j++,
    mutationPair[i, First[lists[[j]]], Last[lists[[j]]]];
    ];
   ];
 ];

(*********************************************************************)
```

```
Print["Number of variants counted in window = ", numVariants];

If[numVariants <= 0, Return[{}]];

(*We now cluster the sequences in the window according to how many
    variants we have counted in the window*)
If[numVariants <= Length[indexedData],
 windowClusters =
   FindClusters[indexedData, numVariants,
    DistanceFunction -> HammingDistance,
    Method -> {"Optimize",
      "Iterations" -> Length[indexedData]}];,
 windowClusters = Map[{Last[#]} &, indexedData];
 ];

Print["Window clusters :", windowClusters];

(*We display the sequences in each derived cluster*)
windowClustersBySeq =
 Map[Map[windowSeq[[#]] &, #] &, windowClusters];

(*We determine the consensus sequence of each cluster (by obtaining the
    majority base for each alignment position given the sequences in a
    cluster)*)
clusterConsensus =
 Map[Map[First[First[Sort[Tally[#], Last[#1] > Last[#2] &]]] &,
    Transpose[#]] &, windowClustersBySeq];

Print["Cluster consensus sequences :", clusterConsensus];
```

```
(*We determine and return the error corrected version of the reads in the
    window*)
correctedReads = Table[If[MemberQ[Flatten[windowClusters], i],
   currentSeq = fullData[[i]];
   pos = First[Flatten[Position[windowClusters, i]]];
   If[splitWindow,
    splitPos = Length[columns] - windowStart + 1;

    StringJoin[
     Table[If[StringTake[fullData[[i]], {j - splitPos}] == "X",
       "X", Take[clusterConsensus[[pos]], {j}]], {j,
       splitPos + 1, Length[clusterConsensus[[pos]]]}],
     StringJoin[
      Table["N", {(windowStart - 1) - (windowEnd2 + 1) + 1}]],
     Table[If[
       StringTake[fullData[[i]], {wLength + j - 1}] == "X", "X",
       Take[clusterConsensus[[pos]], {j}]], {j, 1, splitPos}]],
    StringJoin[
     StringJoin[Table["N", {(windowStart - 1) - 1 + 1}]],
     Table[If[
       StringTake[fullData[[i]], {windowStart + j - 1}] == "X",
       "X", Take[clusterConsensus[[pos]], {j}]], {j, 1,
       Length[clusterConsensus[[pos]]]}],
     StringJoin[
      Table["N", {(Length[columns]) - (windowEnd + 1) + 1}]]]],
   Null], {i, 1, Length[fullData]}];
Print["Corrected version of reads contained in window :",
 Table[i -> fullData[[i]] -> correctedReads[[i]], {i, 1,
   Length[fullData]}]];
```

```
  Return[correctedReads];



  ];

(**************************************************************************)
(*              ERIKSSON ERROR CORRECT MAIN CODE                          *)
(**************************************************************************)
(*Select aligned reads that are comprised by at least two bases*)
alignedReads =
 Select[Table[{indeces[[i]], orig[[i]], alignment[[i]],
    alignment[[i]]}, {i, 1, Length[alignment]}],
  Length[contentPos[Last[#]]] > 0 &];
Print[alignedReads // TableForm];

(*Distinguish leading and trailing gaps in each aligned read from internal
    gaps. Leading and trailing gaps are represented by "X", while internal
    gaps are represneted by "-"*)
alignedReads2 = Table[
  cP = contentPos[Last[alignedReads[[i]]]];
  {First[alignedReads[[i]]], alignedReads[[i, 2]],
   alignedReads[[i, 3]],
   StringJoin[Table["X", {First[cP] - 1}],
    StringTake[Last[alignedReads[[i]]], cP],
    Table["X", {StringLength[Last[alignedReads[[i]]]] -
       Last[cP]}]]}, {i, 1, Length[alignedReads]}
  ];

(*Here we ensure that all the inputted sequencing reads are of the same
    length*)
```

```
max = Max[Map[StringLength[Last[#]] &, alignedReads2]];
 ex = Table[
   max - StringLength[Last[alignedReads2[[i]]]], {i, 1,
    Length[alignedReads2]}];
 alignedReads2 =
  Table[{First[alignedReads2[[i]]], alignedReads2[[i, 2]],
    alignedReads2[[i, 3]],
    StringJoin[Last[alignedReads2[[i]]],
     Table["X", {ex[[i]]}]]}, {i, 1, Length[ex]}];

(*Delete duplicates in list of aligned reads*)
fullData = DeleteDuplicates[alignedReads2, Last[#1] == Last[#2] &];
Print[fullData // TableForm];

orig2 = Map[{First[#], #[[2]], Last[#]} &, fullData];

fullData = Map[Last[#] &, fullData];

(*Index resulting list of aligned reads*)
fullDataIndeces =
 Table[fullData[[i]] -> i, {i, 1, Length[fullData]}];

Print["The full indexed list of aligned reads is: "];
Print[fullDataIndeces];

(*Obtain a list of the columns of the aligned reads*)
columns = Transpose[Map[Characters[#] &, fullData]];

(*Initialise variable to store how many times each column has been
    corrected*)
```

```
columnCoverage = Table[0, {Length[columns]}];

(*Variable to store different window sizes for error correction*)
windowLengthA =
 Select[Table[
   windowLength - i, {i, 0, windowLength, windowOs}], # > 0 &];

(*The algorithm below is run for each window size*)
Table[
 Print["(------------------------------------------------------------)"];
 len = windowLengthA[[k]];
 Print["Window size is now ", len];

 tWR = len*3;
 tWP = 1;

 (*Below we shift three successive windows of size len and perform error
     correction in each of the windows*)

 While[True,

  (*-------------------------------------------------------*)

  (*WINDOW 1*)
  w1p1 = tWP;
  w1p2 = tWP + len - 1;

  (*Condition to terminate error correction - The first window reaches the
      end of the columns*)
  If[w1p1 > Length[columns],
```

```
 Break[];
 ];

(*If the window ranging from column w1p1 to column w1p2 has not been
    evaluated*)
If[! MemberQ[eW, {w1p1, w1p2}],
 Print["Window 1"];

 (*Call windowCorrect to obtain the corrections of reads in this window*)
 (*Append corrected reads to list of corrected reads wC*)
 AppendTo[wC, windowCorrect[w1p1, w1p2, len]];

 (*Append evaluated columns to list of evaluated columns eW*)
 eW = Append[eW, {w1p1, w1p2}];
 ];

(*-------------------------------------------------------*)

(*WINDOW 2*)
w2p1 = tWP + len;
w2p2 = tWP + (2*len) - 1;

If[w2p1 > Length[columns],
 w2p1 = 1;
 w2p2 = len;
 ];

(*If the window ranging from column w2p1 to column w2p2 has not been
    evaluated*)
If[! MemberQ[eW, {w2p1, w2p2}],
```

```
 Print["Window 2"];

 (*Call windowCorrect to obtain the corrections of reads in this window*)
 (*Append corrected reads to list of corrected reads wC*)
 AppendTo[wC, windowCorrect[w2p1, w2p2, len]];

 (*Append evaluated columns to list of evaluated columns eW*)
 eW = Append[eW, {w2p1, w2p2}];
 ];

(*-------------------------------------------------------*)

(*WINDOW 3*)
w3p1 = tWP + (2*len);
w3p2 = tWP + (3*len) - 1;

If[w3p1 > Length[columns],
 w3p1 = 1;
 w3p2 = len;
 ];

(*If the window ranging from column w2p1 to column w2p2 has not been
    evaluated*)
If[! MemberQ[eW, {w3p1, w3p2}],
 Print["Window 3"];

 (*Call windowCorrect to obtain the corrections of reads in this window*)
 (*Append corrected reads to list of corrected reads wC*)
 AppendTo[wC, windowCorrect[w3p1, w3p2, len]];
```

```
   (*Append evaluated columns to list of evaluated columns eW*)
   eW = Append[eW, {w3p1, w3p2}];
   ];

  tWP++;
  ];
 , {k, 1, Length[windowLengthA]}
 ];

(*Print the number of times each position has been corrected*)
Print["Column Coverage :", columnCoverage];



wCT = Transpose[Select[wC, Length[#] > 0 &]];
wCT = Table[{orig2[[i]], wCT[[i]]}, {i, 1, Length[orig2]}];

wCT = Select[wCT, Length[Last[#]] > 0 &];
Print[wCT];
wCT2 = Map[Last[#] &, wCT];

(*We determine the final correction for each aligned read*)
cv = Map[Select[#, StringQ[#] &] &, wCT2];
result = Table[
  If[Length[cv[[j]]] > 0,
   cols = Transpose[Characters[cv[[j]]]];
   seq =
    StringJoin[
     Table[chars =
       Sort[Select[Tally[cols[[i]]], (First[#] != "N") &],
        Last[#1] > Last[#2] &];
```

```
        If[Length[chars] > 0,
         currChar = First[First[chars]];
         If[currChar == "X", "-", currChar]
         , "-"]
        , {i, 1, Length[cols]}]], Null]
     , {j, 1, Length[cv]}];
   result = {Map[First[#] &, wCT], result};
   result = Transpose[Select[Transpose[result], StringQ[Last[#]] &]];
   Return[result];
   ];
```

C.3 QuinceEC Module

```
quinceEC[referenceSequence_, flowgramList_, FVDM_,
   completeLinkageCutOff_] :=
  Module[{flowValueStream, alignmentsWref, flowgramContentPosWref,
    flowgramOverlapIntervalWref, flowgramDistanceWref,
    completeLinkageClustering, clustering,
    flowOrder = Characters["TACG"], generateRandomVariantProportions,
    sequences, pos, sequenceProportions, fm1, fm2, alignments = {},
    alignedFlowgramSeqDist, flowgramContentPos, z, w, clusterSize, U,
    it, score, covergenceTest = {}},


   (*This method returns the alignment position of flowgram i with respect to
      the flow-space representation of the reference sequence*)
   flowgramContentPosWref[i_] :=
    Interval[{First[alignmentsWref[[i]]],
      First[alignmentsWref[[i]]] + Length[flowgramList[[i]]] - 1}];
```

```
(*This method returns the interval of overlap of flowgram i and flowgram j
    given their alignments with respect to the flow-space representation of
    the reference sequence*)
flowgramOverlapIntervalWref[i_, j_] :=
 IntervalIntersection[flowgramContentPosWref[i],
  flowgramContentPosWref[j]];

(*This method returns the distance between flowgram i and flowgram j given
    their alignments with respect to the flow-space representation of the
    reference sequence*)
flowgramDistanceWref[u_, v_] :=
 Module[{li = {}, i, ov, startU, startV, a, p, q, r},
  ov = flowgramOverlapIntervalWref[u, v];
  If[Length[alignmentsWref[[u]]] == 0 ||
    Length[alignmentsWref[[v]]] == 0, Return[-9999]];
  If[Length[ov] == 0, Return[-9999]];
  startU = First[flowgramContentPosWref[u][[1]]];
  startV = First[flowgramContentPosWref[v][[1]]];
  For[i = First[ov[[1]]], i <= Last[ov[[1]]], i++,
   p =
    Total[Table[
      pSgivenN[Last[flowgramList[[u, i - startU + 1]]], n, FVDM]*
       pSgivenN[Last[flowgramList[[v, i - startV + 1]]], n, FVDM]*
       pHomopolymerLengthN[n], {n, 0, 9}]];
   q =
    Total[Table[
      pSgivenN[Last[flowgramList[[u, i - startU + 1]]], n, FVDM]*
       pHomopolymerLengthN[n], {n, 0, 9}]];
   r = Total[
```

```
      Table[pSgivenN[Last[flowgramList[[v, i - startV + 1]]], n,
        FVDM]*pHomopolymerLengthN[n], {n, 0, 9}]];
    a = Divide[p, q*r];
    li = Append[li, a];
    ];
   Return[
    N[Divide[-Log[Fold[Times, 1, li]],
      Last[ov[[1]]] - First[ov[[1]]] + 1]];
   ];

(*This method clusters the flowgrams in flowgramList into j clusters, given
     the alignments of the flowgrams with respect to the flow-space
    representation of the reference sequence*)
completeLinkageClustering[cutOffDistance_] :=
 Module[{m = 0, fmin, fmax, min = -9999, max, clusters, md, r, c,
   re, ce, newCluster, nr},
  clusters = Map[{#} &, Range[Length[flowgramList]]];
  md =
   LowerTriangularize[
    Table[Max[
      Flatten[
       Outer[flowgramDistanceWref, clusters[[i]],
        clusters[[j]]]]], {i, 1, Length[clusters]}, {j, 1,
      Length[clusters]}], -1];
  md = Map[Map[If[# == 0., -9999, #] &, #] &, md];
  fmin = Min[Cases[Flatten[md], Except[-9999]]];
  fmax = Max[Cases[Flatten[md], Except[-9999]]];
  md =
   Map[Map[If[# != -9999,
       Divide[(# - fmin), (fmax - fmin)], -9999] &, #] &, md];
```

```
Print["Clusters initialized to: ", clusters];
While[Length[md] >= 1,
 min = Min[Cases[Flatten[md], Except[-9999]]];
 max = Max[Cases[Flatten[md], Except[-9999]]];
 {r, c} = Sort[First[Position[md, min]]];
 ce = clusters[[c]];
 clusters = Drop[clusters, {c}];
 re = clusters[[r]];
 clusters = Drop[clusters, {r}];
 newCluster = Flatten[{re, ce}];
 nr =
  Table[Max[
    Flatten[
     Outer[flowgramDistanceWref, newCluster,
      clusters[[i]]]]], {i, 1, Length[clusters]}];
 nr =
  Map[If[# != -9999, Divide[(# - fmin), (fmax - fmin)], -9999] &,
    nr];
 AppendTo[clusters, newCluster];
 md = Drop[md, {c}];
 md = Transpose[Drop[Transpose[md], {c}]];
 md = Drop[md, {r}];
 md = Transpose[Drop[Transpose[md], {r}]];
 AppendTo[md, nr];
 md =
  Transpose[Append[Transpose[md], Table[-9999, {Length[md]}]]];
 m++;
 Print["sequence ", m, " clustering: ", clusters, " ", " level=",
   min];
 If[(Abs[min - cutOffDistance] <= 0.05) || (min >=
```

```
        cutOffDistance), Break[];];
    ];
   Return[clusters];
   ];


(*This method returns the alignment position of flowgram i with respect to
    sequence j*)
flowgramContentPos[i_, j_] :=
 Interval[{First[alignments[[j, i]]],
   First[alignments[[j, i]]] + Length[flowgramList[[i]]] - 1}];

(*This method returns the distance between flowgram fI and sequence sJ
    given the alignment of flowgram fI with respect to sequence sJ*)
alignedFlowgramSeqDist[fI_, sJ_] :=
 Module[{fContent, start, end, i},
  fContent = flowgramContentPos[fI, sJ];
  start = First[fContent[[1]]];
  end = Last[fContent[[1]]];
  If[end > Length[sequences[[sJ]]] - 1,
   end = Length[sequences[[sJ]]] - 1];
  Divide[
   Total[Table[-Log[
       pSgivenN[Last[flowgramList[[fI, i - start + 1]]],
        ToExpression[StringTake[sequences[[sJ, i]], -1]],
        FVDM]], {i, start, end}]], end - start + 1]
  ];

(*------------------------------------*)
(*EM ALGORITHM EXPECTATION STEP FORMULA*)
```

```
(*This method determines the conditional probability that sequence j
    generated flowgram i*)
z[i_, j_] :=
 N[Divide[
   sequenceProportions[[
    j]]*(E^-Divide[alignedFlowgramSeqDist[i, j], clusterSize]),
   Total[Table[
    sequenceProportions[[
     k]]*(E^-Divide[alignedFlowgramSeqDist[i, k],
        clusterSize]), {k, 1, Length[clustering]}]]]];
(*--------------------------------------*)


(*This method is used to determine the best underlying homopolymer length
    for sequence j at alignment position l given the z[i,j] matrix*)
U[j_, l_] := Module[{UU, t, ret},
  UU[i_, n_] := Module[{inv, start},
    inv = flowgramContentPos[i, j];
    If[IntervalMemberQ[inv, l],
     start = First[inv[[1]]];
     ret =
      z[i, j]*-Log[
        pSgivenN[Last[flowgramList[[i, l - start + 1]]], n, FVDM]];
     Return[ret];
     ];
    ];

  t =
   Map[Total[Select[#, NumberQ[#] &]] &,
    Table[UU[i, n], {n, 0, 9}, {i, 1, Length[flowgramList]}]];
```

```
  ret =
   StringJoin[flowOrder[[Mod[l - 1, 4] + 1]],
    ToString[First[Flatten[Position[t, Min[t]]]] - 1]];
  Return[ret];
  ];



(*************************************************************************)
(*                QUINCE ERROR CORRECT MAIN CODE                         *)
(*************************************************************************)

(*We obtain a flow-space representation of the reference sequence*)
flowValueStream = flowSpaceRepresentation[referenceSequence];

(*We align each flowgram in flowgramList with respect to flowValueStream
    and select the best alignment position in each case*)
alignmentsWref =
 Map[First[#] &,
  flowgramMatching[flowgramList, flowValueStream, FVDM,
   "SCORE", -1200]];
Print[alignmentsWref];

(*We cluster the flowgrams according to their alignments with respect to
    the reference sequence by complete linkage clustering*)
clustering=completeLinkageClustering[completeLinkageCutOff];

(*We estimate the average cluster size*)
clusterSize = Round[Length[flowgramList]/Length[clustering]];
```

```
(*We initialize j sequences (by copying the flow-space representation of
    the reference genome and introducing random variation)where j is the
    cluster size returned by complete linkage clustering*)
sequences = Table[flowValueStream, {Length[clustering]}];
Table[
 For[w = 1, w < Ceiling[Length[sequences[[j]]]*0.15], w++,
   pos = RandomInteger[{1, Length[sequences[[j]]] - 1}];
   sequences[[j, pos]] =
    StringJoin[StringTake[sequences[[j, pos]], 1],
     ToString[RandomInteger[{1, 5}]]];
   ];, {j, 1, Length[sequences]}];
Print["Clusters initialized(initial random sequences generated) :"];
Print[sequences // TableForm];

(*We initialize the relative proportions of the j sequences*)
sequenceProportions =
 Table[1/Length[clustering], {Length[clustering]}];
Print["New sequence proportions generated : "];
Print[sequenceProportions];

(*We now align each flowgram i with each sequence j*)
alignments =
 Table[Print["Matching flowgrams with sequence ", k];
  fm1 = flowgramMatching[flowgramList, sequences[[k]], FVDM,
    "SCORE", -1200];
  Map[First[#] &, fm1], {k, 1, Length[sequences]}];
Print["Alignments :", alignments];
Print["alignedFlowgramSeqDist :",
 Table[alignedFlowgramSeqDist[p, q], {q, 1, Length[sequences]}, {p,
    1, Length[flowgramList]}]];
```

```
Print["z : ",
 Table[z[p, q], {q, 1, Length[sequences]}, {p, 1,
   Length[flowgramList]}]];

(*We perform 1000 iterations of the Expectation Maximization algorithm*)
For[it = 1, it <= 1000, it++,
 Print["------------------------------------------------------------------"];
 Print["Iteration Number: ", it];

 (*-------------------------------------*)
 (*-------------------------------------*)
 (*EM ALGORITHM MAXIMIZATION STEP*)

 (*1. We derive sequences that will maximize the likelihood of the
    flowgrams given the z[i,j] matrix*)
 sequences =
  Table[U[j, l], {j, 1, Length[clustering]}, {l, 1,
    Length[flowValueStream] + 10}];

 (*Here we introduce some randomness into the derived sequences at
    intervals to ensure the algorithm does not converge to soon*)
 If[((it <= 50) && (Mod[it, 7] == 0)),
  Table[
   For[w = 1, w <= Length[sequences[[j]]]*0.10, w++,
    pos = RandomInteger[{1, Length[sequences[[j]]] - 1}];

    sequences[[j, pos]] =
     StringJoin[StringTake[sequences[[j, pos]], 1],
      ToString[
       Abs[ToExpression[StringTake[sequences[[j, pos]], -1]] -
```

```
          RandomInteger[{1, 5}]]]];
     ];, {j, 1, Length[sequences]}];
 ];
sequences = Map[Append[#, "$"] &, sequences];
Print["New sequences generated :"];
Print[sequences // TableForm];
(*-------------------------------------*)

(*2. We derive relative proportions for the sequences obtained in step 1
    that maximize the likelihood of the flowgrams*)
sequenceProportions =
 Map[Total[#] &,
  Table[z[i, j]/Length[flowgramList], {j, 1,
    Length[clustering]}, {i, 1, Length[flowgramList]}]];
Print["New sequence proportions generated : "];
Print[sequenceProportions];
(*-------------------------------------*)

(*3. We now align each flowgram with respect to each newly derived
    sequence*)
If[it > 5, score = -200, If[it > 3, score = -300, score = -600]];
If[((it <= 50) && (Mod[it, 7] == 0)), score = -1200];
alignments =
 Table[Print["Matching flowgrams with sequence ", k];
  fm1 = flowgramMatching[flowgramList, sequences[[k]], FVDM,
    "SCORE", score];
  Map[First[#] &, fm1], {k, 1, Length[sequences]}];

(*Here we replace some of the alignments with the original alignments of
    the flowgrams with respect to the reference sequence in order to guide
```

```
    the convergence of the Expectation Maximization.
 We do this at intervals of the iteration*)
 If[((it <= 50) && (Mod[it, 8] == 0)),
  Table[
    For[w = 1, w <= Length[alignments[[j]]]*0.10, w++,
      pos = RandomInteger[{1, Length[alignments[[j]]]}];
      alignments[[j]] =
       Join[Take[
         alignments[[j]], {1, pos - 1}], {alignmentsWref[[pos]]},
        Take[alignments[[j]], {pos + 1, Length[alignments[[j]]]}]];
      ];, {j, 1, Length[alignments]}];
  ];
 (*-------------------------------------*)
 (*-------------------------------------*)
 Print["Alignments :", alignments];
 Print["alignedFlowgramSeqDist :",
  Table[alignedFlowgramSeqDist[p, q], {q, 1,
    Length[sequences]}, {p, 1, Length[flowgramList]}]];
 Print["z : ",
  Table[z[p, q], {q, 1, Length[sequences]}, {p, 1,
    Length[flowgramList]}]];


 ];


];
```

Appendix D

List of Reference Sequences used in experiments

We selected HIV pol subsequences that strictly contain the characters {A,C,G,T} as reference sequences. The list of reference sequences used in our experiments is shown below.

1. GenBank: JN570723.1, bases 1 to 150
2. GenBank: JN570723.1, bases 151 to 300
3. GenBank: JN570509.1, bases 1 to 150
4. GenBank: JN570509.1, bases 151 to 300
5. GenBank: JF342314.1, bases 151 to 300
6. GenBank: JF342314.1, bases 301 to 450
7. GenBank: JF342314.1, bases 450 to 600
8. GenBank: JF342314.1, bases 751 to 900
9. GenBank: JF342312.1, bases 151 to 300

D. LIST OF REFERENCE SEQUENCES USED IN EXPERIMENTS

10. GenBank: JF342312.1, bases 300 to 450
11. GenBank: JF342312.1, bases 451 to 600
12. GenBank: JF342312.1, bases 601 to 750
13. GenBank: JF342312.1, bases 751 to 900
14. GenBank: JF342310.1, bases 1 to 150
15. GenBank: JF342310.1, bases 151 to 300
16. GenBank: JF342310.1, bases 301 to 450
17. GenBank: JF342310.1, bases 451 to 600
18. GenBank: JF342310.1, bases 601 to 750
19. GenBank: JF342274.1, bases 1 to 150
20. GenBank: JF342274.1, bases 151 to 300