Fast and Accurate Mapping of Next Generation Sequencing Data

Chandana Tikiri Bandara Tennakoon

(B.Sc.(Hons.), UOP)

A Thesis submitted for the degree of

Doctor of Philosophy

NUS Graduate School for Integrative Sciences and Engineering

National University of Singapore

2013

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Chandana Tikiri Bandara Tennakoon

7th May 2014

Acknowledgements

Starting doctoral studies is like a long journey undertaken by a navigator towards an unknown destination with only a vague sense of direction. The seas are rough and weather can be unpredictable. After five years of journey I have reached the shore. This is how Columbus must have felt when he discovered America.

My journey would have been impossible without the guidance of my supervisor Dr. Wing-Kin Sung. He was my unerring compass. Switching from my background as a mathematics student to computer science went rather smoothly mainly because he identified a suitable topic for me. I am also glad that he emphasized the importance of developing practical tools to be used by bioinformaticians rather than concentrating on toy programs. I am very grateful to him for helping me overcome my financial difficulties and in understanding my family needs. I would also like to thank Prof. Tan Kian Lee and Assoc. Prof. Leong Hon Wei in taking their valuable time to act as my thesis advisory committee members.

Next I would like to thank my ship mates Jing Quan, Rikky, Zhi Zhou, Peiyong, Hoang, Suchee and Hugo Willy. All of your discussions, suggestions and bug reports helped improve my programs immensely. Without Jing Quan and Rikky, I probably would have taken double the time to finish some of my projects. You guys also made the lab a happy place and made me fitter by training with me for the RunNUS. I will miss the fun times for sure. I also would like to thank Pramila, Guoliang, Charlie and Adrianto from GIS for their collaborations.

A sailor cannot start his journey without a ship and provisions. I like to thank NGS for their scholarship and School of Computing for recruiting me as a research assistant. The facilities available at SoC, especially the Tembusu server were excellent. Without the availability of these resources, processing of NGS data would have been impossible.

A journey through unchartered waters is hazardous. Fortunately, pioneering work by Heng Li and the availability of open source software, especially the BWT-SW package which forms a central part in my aligners, guided me immensely. I would also like to thank all the people who disseminate their knowledge in the forums SEQanswers.com and stackoverflow.com free of charge.

Finally I would like to thank my wife and two daughters for their patience. You kept me motivated and happy during hard times.

Contents

Summary

Next Generation Sequencing (NGS) has opened up new possibilities in genomic studies. However, studying the vast amounts of data produced by these technologies present several challenges. In many applications, millions of reads will to be mapped to very large genomes of size around 3 GB. Furthermore, the mapping needs to take into account errors in the form of mismatches and indels.

In this thesis, I introduce fast and accurate techniques to solve NGS mapping problems. Burrows-Wheeler transform (BWT) [20] is a data structure used prominently by sequence aligners. I use BWT based indexing methods to compactly index genomes. My first contribution is a fast and exact method called BatMis [135] to solve the k-mismatch problem. Experiments show that BatMis is more accurate and faster than existing aligners at solving the k-mismatch problem. In some cases, it can produce the exact solution of the k-mismatch problem faster than heuristic methods that produces partial solutions. BatMis can be used to accurately map short reads allowing mismatches [134], and can also be used in pipelines where multiple k-mismatch mappings are required [82, 73].

I next address the problem of mapping reads allowing a mixture of indels and mismatches. This requirement is important to handle longer reads being produced by current sequencing machines. I introduce a novel data structure that can be used to efficiently find all the occurrences of two l-mer patterns within a given distance. With the help of this data structure, I describe an algorithm called BatAlign to align NGS

reads allowing mismatches and indels.

In order to perform accurate and sensitive alignments, BatAlign uses two strategies called reverse-alignment and deep-scan. Reverse-alignment incrementally looks for the most likely alignments of a read, and deep scan looks for hits that are close to the best hits. Finally, the candidate set of hits produced by reverse-alignment and deep scan are examined to determine the best alignment. When handling long reads, BatAlign uses a seed and extend method. I speed up this extension process considerably with the help of a new alignment method and the use of SIMD operations. BatAlign can operate with speeds close to the Bowtie2 aligner which is known for its speed, while producing alignments with quality similar to the BWA-SW aligner which is known for its accuracy.

The last problem I address is mapping RNA-seq reads. I use BatAlign's power to accurately align exonic reads and recover possible junction locations. Furthermore, I use my new data structure to device fast junction finding algorithms. Results from both of these methods are used to determine the best alignment for RNA-seq reads. Furthermore, the algorithm BatRNA uses a set of confident junctions to rectify incorrect alignments and to align junctions having very short overhangs. Comparison with the other state of the art aligners show that BatRNA produces best results in many measures of accuracy and sensitivity, while being very fast.

In summary, the three mapping programs BatMis, BatAlign and BatRNA we present in this theses will provide very attractive solutions to many sequence mapping problems.

List of Abbreviations

BWT	Burrows-Wheeler Transform
BW_T	Burrows-Wheeler Transform of string T
CIGAR	Compact Idiosyncratic Gapped Alignment Record
mapQ	Mapping Quality
NGS	Next Generation Sequencing
RSA	Reduced Suffix Array
SA_T	Suffix array of the string T
SGS	Second Generation Sequencing
SNV	Single Nucleoride Polymorphism
SNV	Single Nucleoride Variation
SW	Smith-Waterman

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

From the time immemorial, people have been seeking answers to questions about life. These questions ranged from those that belonged to the realm of philosophy like "what is the purpose of life" to those that can be treated scientifically like "How did life originate?", "How does life operate?" and "How does life propagate?". Through revolutionary thinking and technological breakthroughs in the last two centuries by people like Darwin, Mendel, Crick, and Watson, the answers to the latter questions have been shown to have a firm molecular basis.

With the publication of "The Origin of Species" in 1859, Charles Darwin initiated a paradigm shift departing from the established view that life on earth was created and is essentially static. He showed that organisms evolve to adapt to the changes in the environment. Later work by Gregor Mendel demonstrated that the propagation of characteristics of a species can be explained in terms of some inheritable factor, which we now refer to as genes. In 1944, Oswald Theodore Avery showed that genetic material is made out of DNA and these series of research finally culminated with the landmark discovery of the double helical structure of DNA by Crick and Watson in 1953.

With the molecular basis of life thus established, scientists became interested in interrogating the structure and the function of DNA. A major breakthrough happened when Maxam, Gilbert [98] and Fred Sanger [126] discovered practical methods to sequence stretches of DNA. This heralded the age of sequencing, and scientists were able to sequence small genomes. In 1977, Sanger himself determined the genome of the bacteriophage OX174 [125] and by 1995, the genome of the first free living organism *Haemophilus influenzae* was completely sequenced [37]. With effective sequencing technologies at hand, scientists launched ambitious projects to sequence the whole genomes of various species having more complex genomes, and to annotate their genes.

These projects, especially the Human Genome Project that was launched in 1990, helped take genomic sequencing to the next level. Due to the large amount of funding pouring in and the competition among laboratories, government agencies, and private entrepreneurs, genome sequencing became much efficient, cheap, and streamlined. Due to this progress, the first draft of the human genome was finished in 2001 [67, 141], two years ahead of its projected finishing date.

Along with the Human Genome, we now have the complete genomes of a wide variety of species publicly available for free. Most of the model organisms like Mouse, Fruit Fly, Zebra Fish, Yeast, *Arabidopsis thaliana, C. elegans* and *E. coli* have been sequenced and their genes have been extensively annotated. Sequencing of well known viruses like HIV (Human Immunodeficiency Virus) or HBV (Hepatitis B Virus), and newly emerging pathogens like SARS (Severe Acute Respiratory Syndrome) virus have also become routine.

1.2 Next Generation Sequencing

Maxam-Gilbert sequencing and Sanger sequencing are called first generation sequencing technologies. Although they were introduced at the same time, Sanger's method was adopted for laboratory and commercial work due to its higher efficiency and lower

radioactivity. Sanger sequencing kept on improving in terms of its cost, ease of use and accuracy. During the Human Genome Project, the sequencing process was parallelized and automated. In 2005, a major improvement in sequencing technologies occurred with the introduction of the 454 sequencer. In a single run, it was able to sequence the genome of *Mycoplasma genitalium* [96]. In 2008, the 454 sequenced the genome of James Watson [148]. The cost and the speed improvements brought forward by 454 were remarkable, and marked the beginning of the Next Generation Sequencing (NGS) technologies, also known as the Second Generation Sequencing (SGS) technologies. Other sequencers competing with 454 appeared within a short time. In 2006, two scientists from Cambridge introduced the Solexa 1G sequencer [11]. The Solexa 1G was able to produce 1GB of sequencing data in a single run for the first time in history. In the same year, another competing sequencer the Agencourts' SOLiD appeared and it too had the ability to sequence a genome as complex as the Human Genome [102]. All these founder companies were acquired by more established companies (454 by Roche, Solexa by Illumina and Agencourt by ABI) and became the major players in SGS.

Newer approaches for sequencing kept on being invented. These include the use of single molecular detection, scanning tunnelling electron microscope (TEM), fluorescence resonance energy transfer (FRET) and protein nanopores [140]. Although there is no accepted categorization, these technologies are sometimes claimed to be the third or fourth generation sequencing technologies [62]. These methods have various advantages and disadvantages compared to each other. Not all of these technologies are fully mature or user friendly; for example, Oxford Nanopore has still not made their sequencer commercially available. Some NGS technologies (e.g. Ion Torrent) are not capable of producing sufficient sequencing coverage for whole genome studies, but are more suitable for clinical applications due to their lower cost, accuracy and faster runtime. Sometimes several sequencers can be used together to take advantage of strengths of each platform. For example, Pacific Bioscience's PacBio sequencer is best used in tandem with other

sequencing platforms. It produces very long reads but the number of reads produced is small. One of its advantages of PacBio is that it does not show much of a sequencing bias, and can be used to sequence regions with high GC content [117].

1.2.1 Algorithmic Challenges of NGS

NGS carries several algorithmic challenges with it. Compared to Sanger sequencing, NGS produces smaller read lengths (though this is bound to change in near future) having more errors. Sequencing methods that amplify and sequence DNA fragments in clusters tend to accumulate errors due to the idiosyncrasies of individual members in the clusters. As the sequencing progresses, these will result in "phasing errors", which causes mismatches to appear (see Chapter 2 for more details). Other methods that sequence individual reads may fail to call bases due to the limits in the sensitivity of measuring devices when homopolymer runs are present. This will result in indel errors. Apart from these, other factors like imperfections in the chemistries will cause sequencing errors too.

Algorithmically, handling exact matches is well studied and many data structures exist to efficiently handle them. However, handling mismatches is not so straightforward, and handling indels is more challenging. While algorithms exist to solve these problems, they tend to be slow. When we take into consideration that millions of reads are produced by NGS, we need to look beyond the classical solutions and towards novel algorithms.

There are other problems associated with NGS. There might be biases in preferentially sequencing regions in genomes, depending on factors like the GC content and the structure of the genome. These biases will result in uneven coverage and can become a problem in the downstream analysis of sequencing data. However, algorithms can employ various methods to compensate for these biases.

1.3 Applications of Sequencing

Compared to Sanger sequencing, NGS technologies produce shorter read lengths. However, they are massively parallel, have higher throughput and are more cost effective. These properties allow NGS to be used in novel ways to unravel mysteries of biology. Some of the highlights of these applications are given below.

1.3.1 *De novo* Assembly of Genomes

In the past decades, Sanger sequencing was the gold standard for *de novo* assembly of genomes due to the long reads it produced. However, Sanger sequencing is costly and time consuming. Due to their high throughput NGS is being successfully employed to assemble genomes. In 2010 BGI assembled the Giant Panda Genome using NGS alone [78], demonstrating the capability of NGS to assemble complex genomes. Also, there are genomes that were assembled using a combination of NGS technologies along with Sanger sequencing [28, 49].

1.3.2 Whole-genome and Targeted Resequencing

Probably the most popular application of NGS is for resequencing. Once a reference genome for a species has been constructed, the whole genome can be sequenced by NGS methods and the generated sequences can be aligned to the reference genome. This would enable the detection of variations like SNPs, indels, copy number variations and structural variations between the sampled genome and the reference genome. These variations can play an important role in the susceptibility to certain diseases; for example, a single nucleotide variation in the gene APOE is associated with a high risk for Alzheimer's disease [149]. Because of the lower cost of NGS sequencing, comparing normal and disease genomes to identify genetic causes for diseases have become increasingly popular. By performing a comparison between the tumor and non-tumor genomes of 88 liver cancer patients, Sung et al. [134] showed HBV integration

in liver cancer patients. Such an effort would have been a very expensive and time consuming proposition before the advent of NGS.

Rather than sequencing a whole genome, a more cost efficient technique is to sequence a targeted region with high coverage. This targeted region could be a gene of interest or the whole exome. Whole exome sequencing was shown to be very effective in identifying Mendelian diseases, when the gene responsible for the extremely rare Miller syndrome was identified using a small number of samples of unrelated individuals [114]. Discovering the genes responsible for such rare inherited diseases would have been a daunting task using traditional methods.

1.3.3 RNA-seq

Detecting the transcripts and the level of their expression is important when understanding the development and disease conditions of a cell. With the introduction of RNA-seq [144], NGS has been increasingly used to study the transcriptomes of cells. The traditional method of using EST has the disadvantages of detecting only about 60% of expressed transcripts and not detecting transcripts with a low level of expression [17]. The large number of reads generated by NGS are more suitable for calculating the expression levels and in detecting rarely expressed transcripts. RNA-seq has advanced transcriptomics by determining the 3' and 5' bounds of genes [112], detecting novel transcribed regions and confirming and detecting novel splicing events [108].

1.3.4 Epigenetic Studies

Epigenetics study the changes in gene functions that do not depend on the DNA sequence. Although sequencing basically determines the linear DNA sequences, it has been successfully used in epigenetics. Two of the areas where NGS is widely used in epigenetics are methylation studies and ChIP-seq.

Methylation of DNA cytosine is an important regulatory process that plays a role

in suppressing gene expression and in cell development. By treating DNA with sodium bisulfite, unmethylated cytosine can be converted to uracil while leaving methylated cytosine intact. When bisulfite treated DNA is sequenced, the unmethylated cytosine will be reported as thymine. Comparing the sequencing data with the reference genome, sites where methylation occurs can be identified [24].

ChIP-seq is a technique that can be used to analyse interaction between proteins and DNA. Using a suitable antibody, sites that interact with the proteins of interest can be pulled down using a method known as chromatin immunoprecipitation (ChIP). These used to be studied with microarrays (ChIP-chip), but NGS has replaced this step [54]. ChIP-seq has been successfully used to study the transcription factor binding sites [54] and Histone modifications [104] and has taken over ChIP-chip as the preferred tool.

1.4 Future of Sequencing

One of the heuristic rules for measuring the success of technological improvements is to see how closely the improvements adhere to Moore's law. i.e. if technological improvements doubles some performance measure every two years, the technology is considered to be doing very well. Judging by this criteria, sequencing technologies are doing exceedingly well. Figure 1.1 shows how the cost of producing a human sized genome has decreased with time. It can be seen that from 2008, when SGS started to replace Sanger sequencing, the improvements have surpassed the level predicted by Moore's law by a wide margin. It shows that within few years, the holy grail of sequencing a human genome for $1000 will be achievable. Once this watershed is passed we can expect routine sequencing of human genomes and their use in diagnostics and personalised medicine. This would be an achievement similar to the discovery of vaccines and antibiotics.

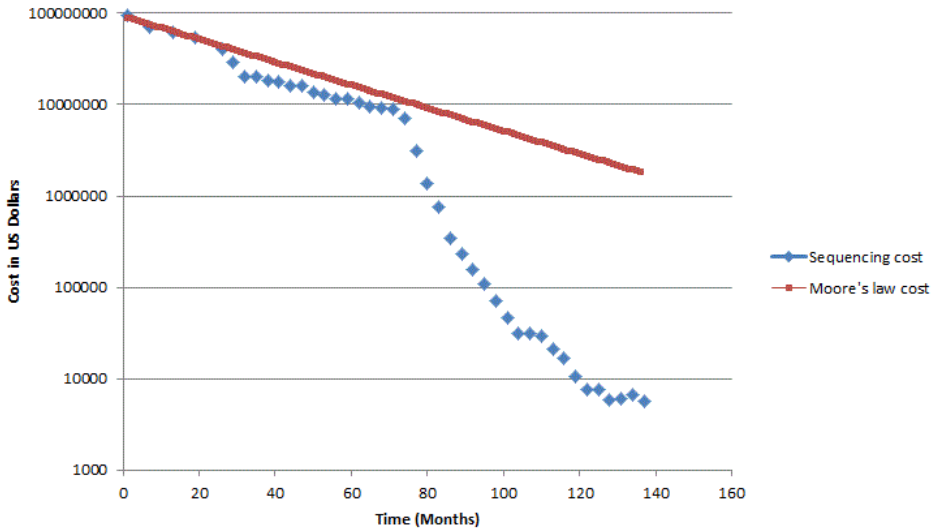


Figure 1.1: *Improvement of the cost to sequence a human sized genome with time. Logarithmic scale is used for the Y axis. Data taken from www.genome.gov/sequencingcosts*

1.5 Aligning NGS Reads

Making sense out of the flood of data generated by NGS requires specialized bioinformatics software. Except for *de novo* assembly, the first step in the applications mentioned above is to align the sequences generated by NGS to a reference genome. Preferably, the aligner will return a mapping quality score indicating the reliability of the alignment. Requirements for the type of alignment differs between applications. Some applications (e.g. SNP calling) choose to use only those reads that can be aligned uniquely with a high mapping quality score, while other applications (e.g. methylation studies, RNA-seq) will use all or the best alignments.

The alignment of NGS reads presents three main difficulties. The first is the size of genomes, the second is the volume of the data generated and the third is the presence of mismatches and indels. An eukaryotic genome like mouse or human can contain around 6GB of DNA if their diplodity is taken into account, or if we consider the similarity of the chromosome pairs, around 3GB of DNA. The sequences generated by NGS differs from the reference genome due to variations between the reference genome and the

sampled genome, and due to errors in sequencing. Therefore, aligning millions of these reads to a genome like the Human Genome presents a huge computational challenge. Classic software like BLAST [7] or BWT-SW [66] are not very useful in this context since they are slow and were not designed with NGS in mind; for example, they do not take into account the nature of sequencing errors and cannot fully utilise additional information like sequencing quality provided by NGS. Therefore, novel approaches are necessary to align NGS reads. The aim of this thesis is the efficient and accurate alignment of NGS reads to a large genome. The following section briefly describes the contributions made by this thesis project in solving this problem.

1.6 Contributions of the Thesis

When aligning NGS reads to a reference genome, two main sources of errors need to be taken into account. They are the mismatch errors and indel errors. Mismatch errors occur when there are SNPs present in the sampled genome, or due to sequencing errors. For NGS sequencers like Illumina and SOLiD, the majority of sequencing errors are of this type. The first contribution of this thesis is the introduction of a fast and memory-efficient algorithm BatMis that can map reads to a genome allowing mismatches. BatMis is an exact method and does not use heuristics. Benchmarks show that BatMis is much faster than other popular aligners at solving the mismatch problem, sometimes even when the other aligners use heuristics. Benchmarks also show that at higher mismatches some aligners might miss correct hits, but BatMis will still return all the correct hits.

When longer reads are considered, indel errors become more important. In order to map reads allowing indels, we introduce the BatAlign algorithm. BatAlign uses the BatMis algorithm, as well as a new data structure I created. Experiments show that BatAlign has better sensitivity and accuracy than other popular aligners.

Finally, we introduce the BatRNA algorithm for RNA-seq mapping. RNA-seq is

harder to handle since its solution needs to take into account the peculiarities of gene splicing. BatRNA can be used to do *de novo* RNA-seq mapping. Again, experiments show that BatRNA is fast, sensitive and accurate. In summary, we present three memory efficient, fast, accurate and sensitive NGS mapping algorithms.

We have published the BatMis algorithm [135], and have used it successfully as an aligner in the whole genome study [134]. Furthermore it provides the multiple mapping in the ChIA-PET (Chromosome Interaction Analysis by Paired-End Tag Sequencing) pipeline [73], and the high mismatch mapping required for the bisulfite read mapper BatMeth [82]. At the time of thesis submission, BatAlign manuscript has been submitted for review, and is undergoing its second round of revision.

1.7 Organization of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2, I present the basic background required for the thesis and a survey of NGS technologies. In Chapter 3, I introduce the FM-index (Full-text index in Minute space) [36] data structure, and describe a new data structure I derived from it. These two data structures will be used extensively in my algorithms. Chapters 4 reviews the NGS mapping algorithms and Chapter 5 reviews the RNA-seq mapping algorithms. Chapter 6 will describe BatMis, my efficient algorithm for solving the mismatch problem. Chapter 7 will describe BatAlign, my algorithm for mapping reads efficiently and accurately allowing mismatches and indels. Chapter 8 will present BatRNA, an algorithm for mapping RNA-seq reads. Finally, Chapter 9 concludes the thesis with a summary of my work and a brief discussion on possible future work.

Chapter 2

Basic Biology and NGS

2.1 Introduction

Living cells can be thought of as complex machines that produce, transform and manipulate various molecules. Capturing and analysing these molecules helps us understand the state of the cells and cellular processes. My thesis concentrates on analyzing sequences of nucleic acids. In this Chapter, I will introduce the structure of nucleic acids and the basic biology behind gene splicing required for the development of the thesis. Furthermore, I will give a brief review of the sequencing technologies.

I would like to note how rapidly the sequencing landscape changes. Between the time of my starting to write the thesis and finishing the final revisions, some platforms have become obsolete and others have gained in prominence. For example, it has been announced that 454 will shut down and the support for the platform will stop in 2016. On the other hand, PacBio has become much more popular. Also, the statistics I collected about sequencers few months ago have changed drastically.

2.2 Nucleic Acids

Nucleic acids can be thought of as the "information macromolecules" of a cell. They are of two types, DNA and RNA. DNA encodes the blueprint for constructing proteins and RNA transfers this information for the assembly of proteins.

2.2.1 DNA

Deoxyribonucleic acid, or DNA, is built by joining together four monomers called nucleotides. The four types of nucleotides are called dAMP, dTMP, dCMP and dGMP. Each of these nucleotides contain a phosphate group, a five-carbon sugar molecule and a nitrogenous base. The five-carbon sugar is deoxyribose. Because the first two components are common to all nucleotides, we identify each nucleotide with their associated nitrogenous base adenine, thymine, cytosine or guanine. They are in turn represented by the initial letters as A,T,C and G respectively.

Two nucleotides can be connected together by joining the phosphate unit of the first nucleotide to the sugar unit of the second nucleotide. Millions of nucleotides can be joined together this way using the sugars and phosphates to form a "backbone"; for example, chromosome 1 of a human contains more than 249 million nucleotides chained together. A long chain of nucleotides formed in this manner is called single stranded DNA or ssDNA. One end of this backbone will have an unbound phosphate unit in the 5' carbon. This end is called the 5' end. The other end of the backbone will have an unbound OH at the 3' carbon. This end is called the 3' end. Therefore, we can assign a sense of direction to a sequence of nucleotides. By convention, sequences of ssDNA are listed from 5' end to the 3' end.

Although ssDNA viruses exist, genomes of higher forms of life consists of double stranded DNA (dsDNA) which we refer to simply as DNA. In DNA, the two strands of ssDNA are joined together by connecting each nitrogenous base in one strand with a nitrogenous base in the other strand. These nitrogenous bases are connected so that

A always pairs with T and C always pairs with G. The pairs A,T and G,C are called complementary pairs.

This describes the linear structure of DNA. However, the real topology of DNA in a cell is more complex. The secondary structure of DNA is the famous double helix. The two strands of DNA run anti-parallel to each other, and are twisted into helices of constant diameter (The sense of direction here is as described previously). In eukaryotic cells, strands of DNA are wound around a family of proteins called histones and can form a complex package.

2.2.2 RNA

RNA is built out of monomers AMP,GMP,UMP and CMP. These monomers are similar in structure to DNA, and consists of a phosphate unit, a five-carbon sugar and a nitrogenous base. However, the five-carbon sugar is a ribose. Instead of thymine a new nitrogenous base uracil, denoted by U, is found. As with DNA, the sugar and phosphate units form a backbone with a similar sense of direction. However, RNA do not form a double helix like DNA. RNA is synthesized using DNA as a template during transcription.

Messenger RNA (mRNA) and transfer RNA (tRNA) are two important types of RNA used in synthesizing proteins. When synthesizing proteins, mRNA is used to obtain a blueprint of the protein and tRNA transports amino acids that make up proteins.

2.3 Genes and Splicing

2.3.1 Genes

The blueprints for the synthesis of molecules required for the function of a cell are encoded in stretches of DNA, and these are called genes. Genes are distributed along

both strands of DNA. Genes consist of transcribed regions and regulatory regions. A transcribed region is converted in to mRNA. The role of the regulatory regions is to mark the location of genes for the transcription mechanism to start and to control the rate of transcription.

The regions in the DNA that corresponds to mRNA are called exons; i.e. exons are the regions in DNA that actually code a protein. In the transcribed region of a gene, the segments lying between exons are called introns. Prokaryotes do not have introns. Eukaryotes need to excise the introns away from the pre-mRNA, and join the exons together. This process is called splicing.

2.3.2 Splicing

The sites where excision of introns are performed are called splice sites. The 5' end of an intron is called a donor site and the 3' end of an intron is called an acceptor site. Splice sites contain special sequences to guide splicing. One of the most conserved signals is the GT at the donor site and AG at the acceptor site. Apart from that, there is a pyrimidine rich region near acceptor sites and most importantly, a branch site where an A is surrounded by some loosely conserved signal sequences. The splicing occurs with the help of a type of RNA-protein complex called snRNP. These bind to the donor site and the branch site, and join together to form a structure called a spliceosome. A lariat is formed by joining the A of the branch site with the donor site. This lariat is cleaved away and the exon at the donor site is then joined with the exon at the acceptor site.

2.3.3 Alternative Splicing

Alternative splicing is the phenomenon of a single gene coding for multiple proteins. This happens during the splicing of of pre-mRNA, and adds to the diversity of proteins. Alternative splicing happens quite often, in fact about 95% of human genes with more than one exon have alternate expressions [116]. The possible number of alternate

splicing can be large too; for example the gene Dscam of *Drosophila melanogaster* can have up to 38,016 possible splicings [127].

Alternative splicing can occur in various manners [13]. One of the most common ways is by skipping an exon. Sometimes splicing occurs in such a way that if one exon is spliced then the other exon will be skipped; i.e. the exons are spliced mutually exclusively. Another possibility is that the donor and acceptor sites might be moved, changing the boundaries of exons. Rarely, an intron may be retained without being spliced out. Furthermore, one or more of the above scenarios can happen at different splice sites, increasing the diversity of alternative splicing even more.

2.4 Sequencing Genomes

The synthesis of macromolecules depend on the structure of DNA in genomes. Therefore, the ability to sequence the genomes provides valuable insights to cellular processes. Chapter 1 gave an overview of the importance and applications of genomic sequencing. We will now present a review of the technologies behind genome sequencing.

2.4.1 Sanger Sequencing

Sanger sequencing uses the idea of chain termination. As described in Section 2.2.1, when forming a chain of nucleotides, the 5' phosphate joins with the 3' OH of the deoxyribose sugar. Dideoxyribose is a sugar identical to deoxyribose except that it has a 3' hydrogen instead of an OH. Now consider a nucleotide whose deoxyribose has been replaced by a dedeoxyribose. Such a nucleotide (called a ddNTP) can be attached to the 3' end of a nucleotide chain, but the chain cannot be extend thereafter due to the lack of OH. This process is called chain termination.

In Sanger sequencing, the DNA are separated into two strands. These seperated strands are put into a mixture having both normal nucleotides and ddNTP, along with a primer designed to bind to the 5' end of one of the strands. When DNA polymerase

is added to this mixture it will start elongating the primer by adding nucleotides from the 5' end to the 3' end. Whenever a ddNTP is added, this elongation process stops. In the next step, the elongated strands will be separated.

The final result is a set of DNA strands of varying lengths having a ddNTP at the 3' end. If they are made to travel through a gel, the shortest (and therefore the lightest) DNA fragments will travel the farthest. This process is known as gel electrophoresis. Therefore, the fragments will cluster and order themselves in the gel according to their size. Modern implementations of Sanger sequencing will add a fluorescent tag to the ddNTP so that each different nucleotide type will emit a different color when they are excited. Finally, the bases are read in the correct order by passing the gel through a laser that excites the fluorescent tags.

2.4.2 Next Generation Sequencing

NGS methods can be categorized into two classes. Sequencers of the first class will amplify the fragments of a genome to form clusters and will sequence these clusters. The other class will try to sequence individual fragments without any amplification. This is called single molecule sequencing (SMS). The first class of sequencers are sometimes classified as the second generation sequencers (SGS) and the other class as the third generation sequencers [85].

The sequencing technologies generally follow some common steps. First the genome is fragmented and custom made DNA segments called adapters are joined to their ends, creating a library. The fragments in the libraries are then attached to a solid base. If the sequencer is clustering based, these fragments undergo amplification. Finally, the bases of each fragment in the library are detected using some mechanism. We will look at how this process works on some of the commercial sequencing platforms.

2.4.3 Roche 454

In the library preparation stage, the DNA will be fragmented and two adapters will be added to their ends. Then the DNA fragments are denatured into two strands, and hybridized onto "capture" beads that have probes complementary to the adapters attached to them. This process is setup to ensure that in the majority of cases, only a single fragment will be attached to a capture bead. Each bead is then enclosed in an oil droplet and subjected to amplification using emulsion PCR [12, 31]. The oil droplet is then dissolved and the beads are transferred to a picotiter plate, which is a large array of small wells designed to trap a single capture bead. Finally each well in the plate is filled with small beads containing enzymes needed for the sequencing reaction.

The sequencing is done by pyrosequencing [119] where light is emitted whenever a nucleotide is added to a chain of nucleotides. A primer is annealed to captured template DNA strands, and each of the four nucleotides are flowed sequentially through the picotiter wells. Starting from the primer, the nucleotides that are added cyclically form complementary pairs with the template, emitting light in the process. The intensity of the light will depend on how many bases are incorporated at a given cycle. This light is captured by a CCD camera and the sequence is interpreted.

The main type of error for the 454 is indels. If a long stretch of the same base, called a homopolymer, occurs in the template sequence the camera might not be sensitive enough to pick up the actual intensity of the emitted light. However, the occurrence of mismatch errors is very low since at a given time only one nucleotide is available for pairing.

2.4.4 Illumina

For Illumina library preparation, DNA is fragmented and a subset of these are selected based on their size. Adapters are then attached to both ends of the fragments. These fragments are added to a glass plate called the flow cell that has probes complementary

to the adaptors, and allowed to hybridize. The free ends of the fragments attach with their complementary adapters on the flow cell creating a bridge shape. Using the adapters as a primer, these structures undergo bridge amplification [3, 35]. After several rounds of bridge amplification, the negative strands are washed away. At this stage the flow cell contains clusters of ssDNA templates.

Illumina uses reversible terminators for sequencing. Similar to Sanger sequencing, this method uses modified nucleotides with the OH in the sugar blocked. However, this block can be removed chemically. Furthermore, the nucleotides have different colored fluorescent tags attached to identify them. The template strands are sequenced by adding a mixture of all four nucleotides, which will result in the incorporation of a single base to the template strands. After washing away the unattached bases, the fluorescent tags are detected. This process is continued after cleaving off the fluorescent tag and unblocking the 3' OH.

The error model of Illumina can be described as predominantly mismatch based and having decreasing accuracy with increasing nucleotide addition steps. The errors accumulate due to failures in cleaving off the fluorescent tags and due to phasing, where bases fail to get incorporated to the template strand or extra bases might get incorporated. Phasing can happen when errors occur in blocking/unblocking of the 3' OH [95].

2.4.5 SOLiD

SOLiD stands for Sequencing by Oligo Ligation Detection. The library preparation is similar to that of 454. The DNA is sheared, separated into single strands and adapters are added to the resulting fragments. These fragments are then captured on beads and amplified using emulsion PCR. The beads are attached to a glass surface on which the sequencing is carried out.

The method of sequencing is called sequencing by ligation [129], and uses 8-mer

probes. The 8-mer probes are fluorescently tagged according to the first two bases of the 8-mer. The sequencing is done in several rounds. First, a primer is attached to all the templates. Then the 8-mers are allowed to hybridise with the template. Those that hybridize adjacent to a primer, or an extension of it, are ligated together. After detecting the color of the fluorescent tag, the added 8-mer is cleaved at the fifth base along with the fluorescent tag and the ligation process continues. When the templates have been sequenced this way, the strands obtained by ligation are washed away and the next sequencing cycle begins, this time adding a primer that is one base off from the previous primer. According to this scheme, after five such rounds of primer resetting and ligation, each base in the template would be interrogated twice by 8-mer probes.

Probing each base twice makes SOLiD's base calling more accurate. Each di-base is encoded using a method called two-base encoding [101], where each di-base is encoded using four different colours. The power of this system lies in its ability to distinguish sequencing errors from genomic variations.

2.4.6 Polonator

Workflow of Polonator is somewhat similar to SOLiD's workflow [107]. The library preparation starts by fragmenting the DNA and circularizing the fragments by joining their ends to the ends of a "linker" DNA. Then the circularized fragments are broken so that the linker is flanked by 17-18bp portions of the original DNA fragments. The result are templates holding two pieces of genomic DNA seperated by a linker. Next, adapters are added to ends of these fragments. Resulting templates are attached to beads, amplified using emulsion PCR and deposited onto a surface.

Sequencing is done using sequencing by ligation. Primers designed to hybridize with adapters holding the 3' and 5' ends of genomic DNA are flowed in. The bases are interrogated by fluorescently tagged 9-mers (called nonamers). A nonamer is designed to determine a specific base only, and the fluorescent tag will correspond to this base. In

one cycle, Polonator adds nonamers designed to probe a specific base away from a primer and lets them hybridize with the templates. Those nonamers that hybridize adjacent to a primer are ligated with the primer, and imaged to determine the attached bases. Next, the primer-nonamer complexes are removed, and another cycle of interrogation is started, this time using nonamers whose query position has been shifted by one base. For both genomic DNA fragments in a template, seven bases are queried away from the 5' end and six bases are queried away from the 3' end. The final result is a 26bp read.

2.4.7 Ion Torrent

Ion Torrent libraries are prepared by shearing the DNA and adding adapters to them. The double strands of DNA are separated and similar to 454, are attached to beads and amplified using emulsion PCR. Primers are added to these beads and the beads are deposited into wells in a silicon chip, called an Ion Chip.

During the sequencing process, the four nucleotides are added sequentially to the wells. The sequencing is done by detecting changes in the pH level as nucleotides are added to a template. Whenever a polymerase adds a nucleotide to a chain, a Hydrogen ion is released, resulting in the change of pH level. The Ion Chip acts as a miniature pH meter, and measures these changes [120].

The main source of error for Ion Torrent is indels. These occur when some templates in a bead fail to incorporate bases in step with others. Another possibility is the failure of pH sensors to correctly measure the change in pH level. Both these errors occur prominently when stretches of homopolymers are sequenced.

2.4.8 HeliScope

HeliScope [136] by Helicos is the first single molecule sequencing system to be developed. In preparing libraries, the DNA is fragmented and a poly-A tail is added to the seperated strands. These are hybridized onto a glass containing poly T probes that will bind to

the poly A tails of the fragments, and will also act as primers.

At the sequencing stage, the templates are washed with polymerase and fluorescently tagged nucleotides. The nucleotides, called virtual terminators, have a special design to prevent the addition of more than one nucleotide at a time [15]. At one cycle, only nucleotides of the same kind are added. The nucleotides that bind with the templates are detected using a sensitive optical system consisting of lasers and a confocal microscope. The next cycle is started after the fluorescent tags and the terminators are cleaved away.

Since the signal from a single fluorescent nucleotide is weak, some bases might not be called. Therefore, the major type of error in HeliScope is deletions [92]. The general error model for HeliScope is predominantly indel based, with substitution errors being rare.

2.4.9 PacBio

PacBio is called single molecule real time (SMRT) sequencing. It does not rely on clusters of templates for sequencing, but detects bases real-time as they are incorporated into a single template. The library preparation start by shearing the DNA and creating what is called an SMRT bell [139] that is different from other library preparation methods in several ways. The DNA double strands are not separated, and the ends of the fragments are capped with two hairpin DNA structures (see Figure 2.1.) These hairpins contain a complementary sequence to a primer. Next, these bell structures are denatured to obtain a circular template.

PacBio uses a technology called zero mode waveguides (ZMW) [71] for sequencing. ZMW's are wells with extremely low volumes, whose diameters are in the order of tens of nanometers. The DNA templates are trapped inside these wells by attaching them to DNA polymerase bound to the bottom of the ZMW. Fluorescently tagged nucleotides enter each ZMW and these are incorporated into the template strands trapped there by the polymerase. In contrast to the usual way of fluorescently tagging the nitrogen

base of a nucleotide, tagging is done to the phosphate of the nucleotide. This aims to make the polymerase action more natural. There is a laser at the bottom of the ZMW to detect the tags. The process of the polymerase adding a nucleotide takes a longer time than an arbitrary nucleotide wandering into the well, and increases the fluorescent signal above background level.

The polymerase used is called a strand-displacing polymerase, and as shown in Figure 2.1 can remove an existing strand from one side while incorporating a new nucleotide on the other side. This allows a circular template to be sequenced continuously. PacBio sequencing is subjected to several forms of errors. A small portion of nucleotides might not get fluorescently tagged properly, resulting in the base not being registered when it is added to the template. There is also the chance of a stray nucleotide remaining too long inside the ZMW, a nucleotide taking longer than average to be added to the template or a fluorescent tag that was clipped not being diffused out of the ZMW quickly, all of which results in an extra base being called. These errors suggests a predominantly indel error model for PacBio. In fact the error rate is extremely high at around 15%. However, by sequencing the circular template several times, a consensus read can be formed that is more than 99% accurate.

2.4.10 Nanopores

Nanopores are holes with a diameter in the order of a nanometer. Some classify nanopore technology as being the fourth generation of sequencing [62]. The detection of bases is done without using any fluorescent tagging, and the use of enzymes is minimal. This reduces the sources of errors common to most platforms discussed above [16]. Properties of nanopores make it possible to detect a change in current when DNA is pulled through a nanopore. Each nucleotide will make a characteristic signal, and this forms the basis for sequencing using nanopores. There are many sequencers being developed using nanopores and nano technology (e.g. Oxford Nanopore, NABsys, Electronic BioSciences,

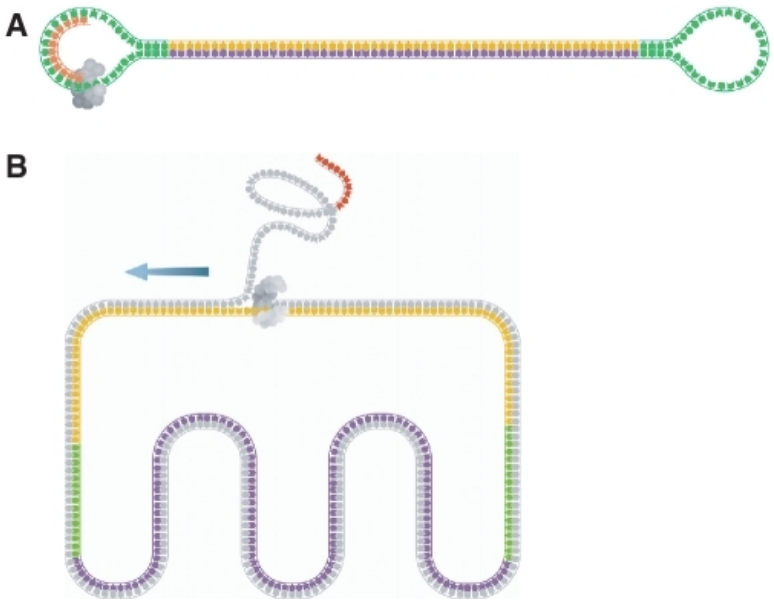


Figure 2.1: *(A) SMRT bell is created by joining two hairpin loops of DNA to a genomic DNA fragment. The hairpin loop has a site for a primer (shown in orange colour) to bind. (B) The SMRT bell is denatured to form a loop, and the strand displacing polymerase (shown in gray) starts adding bases to the loop. When it encounters the primer, it starts displacing the primer and the synthesized strand from one side while adding bases to the strand from the other side. Reproduced from Travers et. al. [139]*

IBM, LongVitae, Complete Genomics, CrackerBio) [154], but Oxford Nanopores is the closest for commercial release.

2.5 SMS vs Non-SMS Sequencing

SMS is more error prone compared to non-SMS due to the weak signals generated when detecting bases. Both types of sequencers are affected by imperfections in their chemical processes and detection methods, but the extent to which they are affected is quite different. If such an error occurs with SMS, a miscall of bases will occur. However, non-SMS methods can still call the correct base as the consensus of a cluster of templates will be taken. On the other hand, errors in SMS tend to be localised but for non-SMS, the errors propagate. For example phasing errors and fluorescent tags that fail to be

removed keep on accumulating, resulting in noisier base calls as read lengths increase. PCR amplification methods show a bias in amplifying certain regions resulting in a bias for non-SMS sequencing. However, SMS are immune to such bias as they will not have an amplification step (except perhaps an initial PCR amplification of the genomic sample).

2.6 Summary

The capabilities of sequencing machines have undergone a rapid evolution in the last decade. However, from an algorithmic point of view, processing the output of sequencing machines pose two distinct challenges; the volume of the data and sequencing errors. The volume of the data will keep on increasing for each platform. Although the sequencing error rate can be expected to go down, it is unlikely that we will ever have a zero error rate. The error profiles of different platforms are different, but would be a mixture of mismatches and indels. When designing good aligners, we should take all these factors into account.

Chapter 3

Burrows-Wheeler Transformation

3.1 Introduction

Searching for occurrences of a substring in a fixed string is an important problem in computer science known as the exact string matching problem. There are two cases of this problem, and they require different approaches. The first case performs the substring search only once, or a limited number of times. Algorithms for solving this problem in linear time have been found [57, 61] and in the average case, even a naive algorithm will solve the problem in linear time. The second case is when multiple substring queries are made. Although the algorithms used for the first case can be used repeatedly for each query, when the number of queries and the sizes of the strings become large, this becomes inefficient. The approach taken in solving this type of problem is to pre-process the fixed string to create a new data structure called an index. Although this one-time indexing operation might be time consuming, it pays dividends as the number of queries increases. In this chapter, we will introduce an indexing data structure called the Burrows-Wheeler transform that can index large strings compactly. With the help of some auxiliary data structures, the exact string matching problem can be solved efficiently.

3.2 Definitions

We define an alphabet Σ to be a finite, non-empty set whose elements are called characters. A string on Σ is defined to be a finite sequence of characters. The empty string is a string having no characters, and is denoted by ε. The set of all strings on Σ is denoted by Σ^*. A string can be represented by writing out its sequence of characters in order. The length of a string S, denoted by $|S|$ is defined as the length of the sequence associated with it. The length of the empty string is 0. A string of length k is called a k-mer.

If $x_i \in \Sigma$ for $i = 1, 2, .., n$, then $S = x_1 x_2 \dots x_n$ would represent a string on Σ and $|S| = n$. We can also use the array notation $S[1..n]$ to indicate a string of length n. With this notation, the i^{th} character of the string is denoted by $S[i]$. If S_1 and S_2 are two strings, the concatenation of S_1 and S_2 is defined to be the string composed of the characters of S_1 followed by the characters of S_2. This is denoted by $S_1 \cdot S_2$ or by $S_1 S_2$. The string obtained by concatenating together n copies of the character x is denoted by x^n. If $S = X \cdot Y \cdot Z$, where X, Y and Z are strings, Y is called a substring of S. The substring of S starting at position i and ending at position j of S is written as $S[i..j]$. When $X = \varepsilon$, Y is called a prefix of S. When $Z = \varepsilon$, Y is called a suffix of S. Non empty prefixes and suffixes are called proper prefixes and proper suffixes respectively. Proper prefixes of S have the form $S[1..j]$, where $1 \leq j \leq n$. Proper suffixes of S have the form $S[i..n]$, where $1 \leq i \leq n$. If $S' = S[i..j]$, then we say that there is an occurrence of S' in S at i.

Two strings S_1 and S_2 are said to be equal if they have the same length and $S_1[i] = S_2[i]$ for all $i = 1, \dots, |S_1|$. Let $<$ be an order relation defined on Σ. Then we can define an order relation $\leq$, called the lexicographical order, on Σ^* as follows. Let S_1 and S_2 be two strings on Σ. Then, $S_1 \leq S_2$ if and only if S_1 is a prefix of S_2 or if S_1 and S_2 can be written as $S_1 = X \cdot a \cdot Y$ and $S_2 = X \cdot b \cdot Y$, where $X, Y \in \Sigma^*, a, b \in \Sigma$ and $a < b$.

3.2.1 Exact String Matching Problem

With the above definitions and notations, we can now formally state the exact string matching problem as follows. Let $T = [1..n]$ and $P[1..m]$ be two strings on an alphabet Σ. The exact string matching problem is to find all the occurrences of P in T. We call T the text and P the pattern.

3.3 Suffix Tries and Suffix Trees

Suffix tree [146] is an indexing data structure that can efficiently solve the exact string matching problem. It is an extremely useful theoretical tool but its practical use is limited due to its large size. When the suffix tree was first introduced, Donald Knuth hailed it as the algorithm of the year, which is a correct description considering its ability to solve common string matching problems elegantly and efficiently. We will now look at a related data structure called the suffix trie from which the suffix tree is derived. For the rest of the chapter, we will assume that the text T to be indexed is a string on a special alphabet with the order relation "$<$" containing a special terminator character "\$". The terminator character appears only once in T as the final character. The reason for this choice will be explained later.

Suffix trie for T is defined to be a rooted tree with the following properties.

1. The tree has exactly $|T|$ nodes.
2. Each edge is labelled with a character from Σ.
3. Concatenating labels on edges of every path from the root to a node will produce all the suffixes of T.
4. Labels of edges starting at each internal node will not contain the same character twice.

Note that unless there is a special terminating symbol "$" for T, if a substring of T occurs at two distinct locations, condition 3 cannot be satisfied. We define threading of pattern P through the suffix trie of T to be the matching of characters of P from left to right in the unique path along T until all characters in P are matched or no more characters of P can be matched [56]. If threading stops due to the former reason, we say it is a complete threading. Otherwise the threading is said to be incomplete. Every complete thread stops at a vertex of the suffix trie. We define P-matching leaves to be the leaves of the subtree rooted at such a vertex.

3.3.1 Solution to the Exact String Matching Problem

Once we constructed the suffix trie of T, the exact string matching problem can be easily solved. The algorithm is given in Algorithm 3.1. Since there are at most $|\Sigma|$ edges per node, the worst case time to thread a pattern P through the suffix trie of T is $O(|P|)$, and if there are *occ* occurrences of P in T, the whole process to report the occurrences will take $O(|P| + occ)$ time.

Algorithm 3.1: Exactmatch($Suff\_T, P$)

```
Data: Suff_T is the suffix trie of string T and P is a string
Result: Find all occurrences of P in T
1 Thread P through Suff_T;
2 if threading is complete then
3     report all positions of P-matching leaves;
4 end
```

3.3.2 Suffix Trees

Suffix trie requires $O(n^2)$ space in the worst cases (for example, when $T = a^n b^n$ for two distinct characters a and b). We can compact the suffix trie by removing the internal nodes that have only one outgoing edge. Then, the edges between the remaining nodes can be concatenated and labelled with the string obtained by concatenating the labels

of the consecutive edges. Now, each internal node will have at least two successors and since there are exactly n leaves, the number of internal nodes will not exceed $n-1$. Therefore, the total number of edges will be $O(n)$.

Next, we will find a more succinct labeling scheme for the edges. Since the edges are labelled by a substring $T[i,j]$ of T, we can relabel them by the ordered pair (i,j). Each such label will cost $2\log(n)$ bit-space. Therefore, the final space complexity for the new compacted suffix trie, which is called a suffix tree, is $O(n\log(n))$ bits.

Construction of suffix trees has been widely studied. Earliest algorithms were able to construct suffix trees in linear time [147] using a large amount of memory, but [99] improved the space complexity to $O(n^2)$. The algorithm by Farach [33] is a popular algorithm that has been adopted to construct suffix trees and its variants.

The definition of threading through a suffix trie in Section 3.3 and Algorithm 3.1 can be easily carried over to suffix trees. Although suffix tree is a better alternative to the suffix tries, their $O(n\log n)$ space requirement is still quite prohibitive for indexing large genomes. In the next section, we will describe another data structure that can index a text more compactly.

3.4 Suffix Array

Suffix array was introduced in 1993 by Manber and Myers [93]. The suffix array $SA_T[1..n]$ of a text T is an array containing the suffixes of T sorted in the lexicographically increasing order. Instead of storing the suffix itself, the position of the suffix is stored. Therefore, $SA_T[1..n]$ is an integer array such that $T[SA_T[i]..n]$ is the i^{th} smallest suffix in T. The space required to store each position is $O(\log n)$ bits, so the space complexity of this data structure is $O(n\log n)$ bits.

Since the suffix array stores the suffixes in lexicographical order, all the suffixes of T that have P as their prefix will be found consecutively. Therefore, if $SA[i]$ and $SA[j]$ are the lexicographically smallest and largest suffixes of T having P as a prefix, then

$SA_T[i], \ldots, SA_T[j]$ are the locations where P occurs as a substring of T. This interval $[i, j]$ is called the SA_T-range of P. $SA_T[i]$ is called the SA_T-value of i, and i is called an SA_T-location.

Given a suffix trie/tree, the corresponding suffix array can be easily obtained by performing a depth-first traversal in lexicographical order and reporting the positions of suffixes at the leaves. However there are more efficient methods of constructing the suffix array in linear space and time [46].

3.4.1 Exact String Matching with Suffix Array

The exact string matching with suffix arrays is done with a binary search, and the algorithm is given in Algorithm 3.2. In the algorithm, the variables L and R are top and bottom bounds of $SA_T[1..n]$ where P might occur as a suffix. They are initially set to the start and the end of $SA_T[1..n]$. The algorithm compares the suffix corresponding to the SA_T-value in the middle of L and R with P. If there is a match it is reported. Otherwise the bounds L and R are narrowed down. This process terminates when $L = R$ (which indicates that P occurs in T) or when $L > R$ (which indicates that P does not occur in T.)

Each iteration of the loop in the algorithm halves the distance between L and R at Line 3. Therefore, the algorithm will iterate at most $\log n$ times. Finding the longest common prefix in Lines 4 takes $O(m)$ time, and the time complexity of the algorithm is $O(m \log n)$. However, using an additional data structure called a longest common prefix array, the time spent on comparing P with the prefixes in $SA_T[1..n]$ can be reduced. This will result in an algorithm that has the time complexity $O(|P| + \log n + occ)$ [93].

Although suffix arrays offer better space reduction compared to the suffix tries, they do so by trading off speed. There are other variants of suffix array that reduces the space even more [42]. Also, there are some software that use suffix arrays to index large genomes [30, 44] despite their large size. However, the space requirement is still too

Algorithm 3.2: Exactmatch_SA($SA_T[1..n], P$)

Data: $SA_T[1..n]$ is the suffix array of string T and P is a string
Result: Find all occurrences of P in T

```
1  L = 1, R = n;
2  while L ≤ R do
3      M = ⌊L + R⌋/2;
4      if Longest common prefix of P and SA_T[M] = P then
5          Report SA_T[M];
6      else if T[SA_T[M]..n] > P then
7          R = M;
8      else
9          L = M;
10     end
11 end
```

steep for many users. The next index we present actually manages to reduce the index size to the same size as the original text.

3.5 The Burrows-Wheeler Transform

The Burrows-Wheeler transform (BWT) was first discovered as a method of compression [20]. However, it proved to be extremely useful as an indexing method. BWT is simply an easy to invert permutation of a string. We will denote the BWT of T by $BW_T[1..n]$, and it can be defined using the suffix array as

$$BW_T[i] = \begin{cases} T[n] & \text{if } SA_T[i] = 1 \\ T[SA_T[i] - 1] & \text{otherwise.} \end{cases}$$

i.e. it is obtained by concatenating the characters preceding the sorted suffixes. Data compression-wise, this leads to a string amicable to compression as the context of consecutive entries of the suffix array tends to be similar, and the chance of similar characters preceding them is also high. Next we will show how to invert the $BW_T[1..n]$ to obtain T. Table 3.1 shows an example of $BW_T[1..n]$ and $SA_T[1..n]$ for a string T.

BWT	Suffix Array	Sorted suffixes
g	7	$
$	1	acacag$
c	3	acag$
c	5	ag$
a	2	cacag$
a	4	cag$
a	6	g$

Table 3.1: *The BWT and the suffix array along with the sorted suffixes of string acacag$. Note that the BWT of string can be easily compressed.*

Let $occ(x, i)$ be the number of occurrences of x in $BW_T[1..i]$, and let C be an array such that $C[x]$ is the number of characters in $BW_T[1..n]$ that are smaller than x for all $x \in \Sigma$. Then we have the following lemmas.

Lemma 3.5.1. *Let $a = BW_T[i]$, where $1 \leq i \leq n$. Then, among the suffixes that start with a, there are $occ(a, i-1)$ suffixes lexicographically smaller than $a \cdot T[SA_T[i]..n]$ and $occ(a, i)$ suffixes lexicographically smaller than or equal to $a \cdot T[SA_T[i]..n]$.*

Lemma 3.5.2. *Let $a = BW_T[i]$, where $1 \leq i \leq n$. If $SA_T[i] \neq 1$, then there are $C[a] + occ(a, i-1)$ suffixes smaller than $T[SA_T[i]-1..n]$, and $C[a] + occ(a, i)$ suffixes smaller than or equal to $T[SA_T[i]-1..n]$.*

Proof. All suffixes that start with $a' < a$, where $a' \in \Sigma$, are smaller than $a \cdot T[SA_T[i]..n]$. From Lemma 3.5.1, there are $occ(a, i)$ suffixes smaller than $a \cdot T[SA_T[i]..n]$ that start with a. Adding them up, and noting that $BW_T[i] = T[SA_T[i]-1]$, we get the required result. □

Lemma 3.5.3. *Let $1 \leq i \leq n$. If $SA_T[i] \neq 1$ and $a \cdot T[SA_T[i]..n]$ is a substring of T, then there are $C[a]+occ(a, i-1)$ suffixes smaller than $a \cdot T[SA_T[i]..n]$, and $C[a]+occ(a, i)$ suffixes smaller than or equal to $a \cdot T[SA_T[i]..n]$.*

Proof. This is a re-statement of Lemma 3.5.2. □

Lemma 3.5.4 (Inversion of BWT)**.** *Let* $1 \leq i \leq n$*. If* $T[j] = BW_T[i]$*, then* $T[j-1] = BW_T[C[T[j]] + occ(T[j], j)]$ *for* $j > 1$*.*

Proof. Since $BW_T[i] = T[SA_T[i] - 1]$ by definition, we have $T[SA_T[i] - 1..n] = T[j..n]$. From Lemma 3.5.2, the $SA_T[1..n]$ position of $T[SA_T[i] - 1..n]$ is $C[T[j]] + occ(T[j], i)$. Therefore, $T[SA_T[C[T[j]] + occ(T[j], i)..n] = T[j..n]$. By definition, $BW_T[C[T[j]] + occ(T[j], i)] = T[j-1]$. □

Algorithm 3.3: Invert_BWT($BW_T[1..n], k$)

Data: $BW_T[1..n]$ is the BWT of string T, $BW_T[k]$ is the terminal character of Σ.
Result: Find T

```
1 i = k;
2 for j = n to 1 do
3     T'[j] = BW_T[i];
4     i = C[T'[j]] + occ(BW_T[T'[j]], i);
5 end
6 report reverse of T'[1..n].
```

Algorithm 3.3 gives the algorithm to invert $BW_T[1..n]$. k is the place where the terminal character of T has been mapped to $BW_T[1..n]$ under the BWT. Starting from this last character of T, the loop in the algorithm takes the location of the j^{th} character of T in $BW_T[1..n]$ and computes the location of $(j-1)^{\text{th}}$ character of T in $BW_T[1..n]$ by applying Lemma 3.5.4. This will reconstruct T in the reverse order. Line 6 rectifies this by reporting the reverse of the reconstructed string.

Note that if we start with k at a location different from the end, the algorithm will still start inverting the BWT from that point backwards. If a data structure is available to compute $C[a]$ and $occ(a, i)$ in constant time, the BWT inversion algorithm can be run in linear time. The next section discusses how to construct such data structures.

3.6 FM-Index

FM-index, which stands for full-text index in minute space, is a data structure proposed by Ferragina and Manzini [36] that indexes a compressed text. Since genomes do not tend to compress well, we will discuss how this data structure is implemented on an uncompressed text. The FM-index consists of $BW_T[1..n]$ and some auxiliary structures to compute C and $occ(a, i)$.The space required for the index is $O(n)$-bits. This is a significant advantage over the indexing methods we have discussed so far that require a much larger space than the original text. To index a nearly 3 GB genome, a suffix tree takes up about 40 GB of space while a suffix array takes up about 12 GB. However, the FM-index takes up only about 750 MB.

3.6.1 Auxiliary Data Structures

The array C can be computed by counting the occurrences of each character, and will take $O(|\Sigma|)$ space. The difficulty is in designing a succinct data structure to compute $occ(a, i)$. We first partition $BW_T[1..n]$ into "buckets" of size l^2, where $l^2 \leq n$. There will be $\frac{n}{l^2}$ such buckets. Each of these buckets are further divided into sub-buckets of size l. For each $a \in \Sigma$, we store the total number of a in buckets and sub-buckets as follows. For the i^{th} bucket, we store the count of a's in $BW_T[1..il]$, denoted by $B_a[i]$. For the j^{th} sub-bucket in bucket i, we store the number of a's in the first j buckets, denoted by $B'_a[i, j]$. Finally, for all $i \leq l$, we create a lookup table $R_a(S, c)$ that returns the number of occurrences of a in the first i characters of a string S of length c, where $1 < c < \log n$ is an integer.

With these counting data structures, $occ(a, i)$ can be easily calculated as follows. i falls into the $\lfloor \frac{i}{l^2} \rfloor^{\text{th}}$ bucket and the $\lfloor \frac{i}{l} \rfloor^{\text{th}}$ sub-bucket, and the last $i - \lfloor \frac{i}{l^2} \rfloor - \lfloor \frac{i}{l} \rfloor$ characters will not fall completely in any sub-bucket. The number of a's in $BW_T[1..i - \lfloor \frac{i}{l^2} \rfloor - \lfloor \frac{i}{l} \rfloor]$ is $B_s[\lfloor \frac{i}{l^2} \rfloor] + B'_a[\lfloor \frac{i}{l^2} \rfloor, \lfloor \frac{i}{l} \rfloor]$. If $BW_T[1..i]$ does not fall completely in to a sub-bucket, then the a's in $BW_T[i - \lfloor \frac{i}{l^2} \rfloor + \lfloor \frac{i}{l} \rfloor..i]$ needs to be counted. This segment can be divided

in to consecutive intervals of length c and the number of a's in each segment can be counted using R_a. Adding all these counts of a will give $occ(a, i)$.

Calculation of $occ(a, i)$ using B, B' and R_a involves only table lookups, and can be done in constant time. If we set $l = \log n/c$ then the space complexity of these structures can be calculated as follows. B stores $O(n/\log^2 n)$ integers ranging from 1 to n. Since these integers can be represented with $\log n$ bits, B can be stored in $(n/\log n)$ bit-space. B' will require $O(n/\log n)$ entries taking $O(\log\log n)$-bit space, resulting in $O(\frac{n\log\log n}{\log n})$-bit space. The final table R_a requires $O(2^{\log n/c}\log n)$ entries of $O(\log\log n)$-bits, and needs $O(n^{\frac{1}{c}}\log n\log\log n)$, which is $O(n)$-bit space. Therefore, the final space complexity of these data structures is $O(\frac{n\log\log n}{\log n})$-bits.

3.6.2 Exact String Matching with the FM-index

We will next present the solution to the exact string match problem using an FM-index. The algorithm is shown in Algorithm 3.4 and is called the backward search. The algorithm uses the following lemma;

Lemma 3.6.1. *Let $[p, q]$ be the SA_T-range of S. Then the SA_T-range of aS is $[C[a] + occ(a, p-1) + 1, C[a] + occ(a, q)]$.*

Proof. From Lemma 3.5.3, the number of suffixes lexicographically smaller than $aT[SA[p]..n]$ is $C[a] + occ(a, p-1)$. Therefore, $C[a] + occ(a, p-1) + 1$ suffixes are lexicographically smaller than or equal to $aT[SA[p]..n]$. Again by Lemma 3.5.3, the number of suffixes lexicographically smaller than or equal to $aT[SA[q]..n]$ is $C[a] + occ(a, q)$. Noting that $T[SA[p]..n]$ and $T[SA[q]..n]$ are the lexicographically smallest and largest substrings in T that have S as a prefix, we get that SA_T-range of aS is $[C[a] + occ(a, p-1) + 1, C[a] + occ(a, q)]$. □

In the algorithm, $[st, ed]$ is an SA_T-range that is initialized to the SA_T-range of $P[|P|]$ in Line 1. In the while loop, $[st, ed]$ will be the SA_T-range of $P[i..l]$. The loop

will take the SA_T-range of $P[i+1..|P|]$ and find the SA_T-range of $P[i+1..|P|]$ using Lemma 3.6.1. If $st < ed$, the resulting SA_T-range is invalid and P is not a substring of T. Otherwise, $[st, ed]$ will be the SA_T-range corresponding to P.

Lines 4-7 of the algorithm can be performed in linear time, and the loop will be executed at most $|P|$ time. Therefore, the time complexity of the algorithm is $O(|P|)$.

Algorithm 3.4: Backward_Search($BW_T[1..n], P$)

Data: $BW_T[1..n]$ is the BWT of string T and P is a string
Result: Find SA_T range of P in T, and the number of occurrences of P in T.

```
1  x = P[|P|]; st = C[x] + 1; ed = C[x + 1];
2  i = m − 1;
3  while st ≤ ed and i ≥ 1 do
4      x = P[i];
5      st = C[x] + occ(x, st − 1) + 1;
6      ed = C[x] + occ(x, ed);
7      i = i − 1;
8  end
9  if st ≤ ed then
10     report [st, ed];
11     report number of occurrences ed − st;
12 end
13 report zero occurrences;
```

3.6.3 Converting SA_T-Ranges to Locations

The result of the backward search algorithm is an SA_T-range, and not the locations of the occurrences of P in T. To convert an SA_T-range $[st, ed]$ into the corresponding locations, we need to take each $i \in [st, ed]$ and find their SA_T-values. We will call this operation as decoding, and it can be easily done if $SA_T[1..n]$ is available. However, storing $SA_T[1..n]$ defeats the purpose of constructing the FM-index to save space. Given a fixed $k \geq 0$ and any $\epsilon > 0$, [36] describes a data structure that occupies $O(H_k(T) + \frac{\log\log n}{\log^\epsilon n})$-bit space and retrieves the SA_T-value corresponding to any SA_T-location in $O(\log^\epsilon)$ time, where $H_k(T)$ is the k-$^{\text{th}}$ order empirical entropy of T. We will

now describe a practical method that can decode any SA_T-location in $O(\log n)$ average time while taking only $O(n)$-bit space.

The algorithm is described in Algorithm 3.5. We sample the values of SA_T at fixed intervals and store them in an array, i.e., we store $D_\kappa[i] = SA[\kappa i]$ for some $\kappa = O(\log n)$ and $i = 1, 2, ..\lfloor n/\kappa \rfloor$. D_κ will then take up $O(n)$-bit space. If we want to decode an SA_T-location s and if $s = \kappa j$ for some $j = 1, 2, ..$, then $D_\kappa[s]$ would give the corresponding location. Otherwise we use algorithm 3.3 to invert the BWT starting from $BW_T[s]$ until we come across a value $BW_T[\kappa j]$. If this value is reached after v steps, the location corresponding to s is $D_\kappa[\kappa j] - v$. Since SA_T is uniformly sampled at intervals of length $O(\log n)$, the average value of v is $O(\log n)$. Therefore, s can be decoded in $O(\log n)$ time.

Algorithm 3.5: Decode$(s, D_\kappa, BW_T[1..n])$

Data: s is an SA_T-location, $BW_T[1..n]$ and D_κ is a sampling of $SA_T[1..n]$ at uniform intervals of κ
Result: Find SA_T-value of s

```
1 s' = s; v = 0;
2 while s' ≠ κj do
3     c = BW_T[i];
4     s' = C[c] + occ(BW_T[c], i') ;          /* invert the BWT */
5     v = v + 1;
6 end
7 return D_κ[s'] − v;
```

The algorithms in sections 3.5-3.6 are based on the work of Ferragina et. al. [36].

3.7 Improving Decoding

Although the decoding operation can be performed in $O(\log n)$ time, practical applications may require thousands of decoding operations to be performed per string search. Therefore, decoding may become a bottleneck in some applications. In the following sections, I will describe a novel data structure I invented and some observations that can

be used to make the decoding operations faster. First, I will describe how to efficiently find the location of a pattern on the fly while performing a backward search with the following lemma.

Lemma 3.7.1. *Let P be a substring of T. When performing backward search for P, if the SA_T-range corresponding to $P[l, |P|]$ is in the form $[\kappa i, \kappa i]$, Then P occurs uniquely at the location $SA[\kappa i] - (l - 1)$.*

Proof. The unique suffix corresponding to $[\kappa i, \kappa i]$ is $SA_T[\kappa i]$, and we have $P[l, |P|] = T[SA_T[\kappa i], SA_T[\kappa i] + |P| - l]$. Therefore, P occurs uniquely at $SA[\kappa i] - (l - 1)$. □

Since $SA_T[\kappa i]$ is already sampled, if any backward search satisfies this condition, we can obtain the location of P in constant time as soon as the search terminates. As P becomes longer, we can expect P to occur uniquely in T most of the time, and to satisfy the above condition.

3.7.1 Retrieving Hits for a Fixed Length Pattern

It is common to search for patterns of fixed length when mapping reads. We will now describe a simple but powerful data structure that will speedup the retrieval of such hits. This data structure is based on the observation that, when the length of the pattern l is long, the corresponding SA_T-range of the pattern is usually small. In fact, assuming that genomic sequences are random strings, we can show that the expected size of the SA_T-range for a length-$(\log_{|\Sigma|} n)$ pattern is 1. On the other hand, if the SA_T-range of a length l pattern is too long, the pattern is more likely to be from low complexity or repeat regions. Such patterns tend to be noisy. By ignoring these types of hits, we might be able to increase the specificity [115] while making an acceptable trade off with sensitivity.

Based on these two observations, our data structure stores the SA_T-values for patterns whose SA_T-ranges have a size between δ and D ($\delta < D$) as follows. Let

$\mathcal{P} = \{P_1, \ldots, P_r\}$ be the set of distinct length-l substrings of T whose number of hits in T are between δ and D. Let $[s_k, e_k]$ be the SA_T-ranges of P_k for $k = 1, 2, \ldots, r$. Note that the SA_T-ranges of all these patterns are distinct and do not overlap. We define the reduced suffix array RSA_T to be an integer array formed by retaining only the SA_T-values of $[s_k, e_k]$ for all $k = 1, \ldots, r$. RSA_T stores entries for $[s_k, e_k]$ consecutively for $k = 1, \ldots, r$ and the SA_T-values belonging to each $[s_k, e_k]$ are kept sorted in their positional order. Furthermore, we also store two arrays $S[1..r]$ and $SUM[1..r]$ where $S[k] = s_k$ and $SUM[k] = \sum_{i=1}^{k-1}(e_i - s_i + 1) + 1$. $SUM[k]$ will point to the beginning of the locations corresponding to $[s_k, e_k]$. We denote the data structure consisting of the BWT of T, the uniform sampling of $SA_T[1..n]$ at κ interval D_κ, RSA_T, S and SUM by $L_{T,l,\delta,D,\kappa}$.

Figure 3.1 shows an example of this data structure. Consider the string "$T =$ *cccctgcggggccg*\$". This string contains the 2-mers cc, cg, ct, gc, gg and tg. The BWT of T is "*tccgccgc*\$*gggccg*". Only the 2-mers cc, gg, gc and cg appear more than once in T. Figure 3.1(a) lists down the occurrences of each of these 2-mers in T sorted by their location, along with the corresponding SA_T-ranges. To construct $L_{T,2,2,3,\kappa}$, we need to consider all 2-mers that appear at least twice and at most three times. In Figure 3.1, the SA_T-ranges are sorted by their starting positions. Then, these starting positions are used as keys pointing to the beginning of the list of sorted locations belonging to the corresponding SA_T-ranges. If we are to construct $L_{T,2,2,4,\kappa}$, then the entry for cc will be included as it occurs 4 times in T.

Table 3.2 shows that $L_{T,l,\delta,D,\kappa}$ is space-efficient in practice, for a large genome like hg19. It gives the space requirement for the data structure, excluding the space occupied by $BW_T[1..n]$ and D_κ, and for $\delta = 4$ and $D = 30,000$.

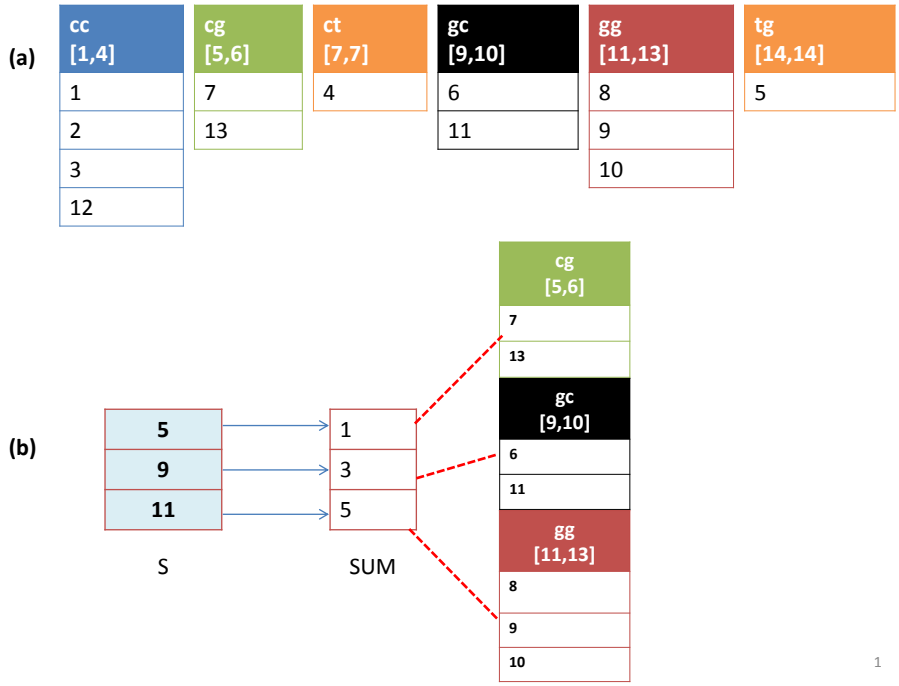


Figure 3.1: *Illustration of the data structure for fast decoding of SA_T-ranges. cccctgcggggccg\$ gives an example string T. (a) shows the sorted positions of every non-unique 2-mer in the string. The label on top of of each list indicates the 2-mer and its SA_T-range. (b) Data structure $L_{T,2,2,3,\kappa}$ that indexes each 2-mer by the starting position of each SA_T-Range. The 2-mers ct and tg are not indexed as they occur uniquely in T. cc is not included as it has four occurrences.*

l-mer size	Size of $L_{T,l,\delta,D,\kappa}$
18	2.5 GB
25	1.6 GB
30	495 MB
37	361 MB
68	244 MB
75	199 MB
90	140 MB
100	116 MB

Table 3.2: *Size of the data structure $L_{T,l,\delta,D,\kappa}$ for different values of l, where T is the hg19 genome, $\delta = 4$ and $D = 30,000$. The size excludes the sizes of $BW_T[1..n]$ and D_κ*

3.8 Fast Decoding

Now, given $L_{T,l,\delta,D,\kappa}$ and the SA_T-range $[i, j]$ of a pattern P of length l, we can use the algorithm *SA_to_Loc* in Algorithm 3.6 to get all of its occurrences in T efficiently. If P occurs at most δ times, or more than D times, we use the method given in Algorithm3.5 to obtain each $SA_T[i], \ldots, SA_T[j]$ (Lines 9-14). Otherwise, we use $L_{T,l,\delta,D,\kappa}$ to find all the occurrences as follows. First, the array S containing the starts of SA_T-ranges is binary searched for i. If $S[k]$ contains i, then $SUM[k]$ will contain the position of the RSA_T containing all the SA_T-values of $[i, j]$ (Line 4).

The time taken for decoding an SA_T-range of a pattern P is $O(occ)$ if the number of occurrences of P in T is larger than δ but does not exceed D. Otherwise, it will take an average time $O(occ \log n)$. However, in practice, we will discard patterns with $occ > D$, and will be using the decoding method described by Lemma 3.7.1 to decode unique reads. Therefore, for sufficiently long P, we would be able to decode SA_T-ranges in an $O(occ)$ average time using the new data structures.

Algorithm 3.6: SA_To_Loc($[i, j], L_{T,l,\delta,D,\kappa}$)

Data: $[i, j]$ is an SA_T interval
Result: Find all locations corresponding to $[i, j]$

```
1  if (j − i) > δ and (j − i) ≤ D then
2      Binary search S for the key i;
3      if S[k] = i then
4          return RSA_T[SUM[k]], ..., RSA_T[SUM[k + (j − i)] − 1];
5      else
6          return ∅;
7      end
8  else
9      H = ∅;
10     for k = i, .., j do
11         L = Decode(k, D_κ, BW_T[1..n]);
12         H = H ∪ L;
13     end
14     return H;
15 end
```

3.9 Relationship Between Suffix Trie and Other Indices

The relationship between the suffix trie and the suffix tree is clear. Each node in the suffix tree corresponds to a node in the suffix trie, and the internal nodes with one outgoing edge in the suffix trie will be absent in the suffix tree. Let the suffix tree be constructed so that the edge labels produced in a breadth first search at each node will always be lexicographically sorted in the descending order. If we now do a depth first search, each leaf in the suffix tree will correspond to the SA-values from bottom to the top. Now consider an internal node, and let P be the string obtained by concatenating the edge labels from the path from the root to the node. Then, the subtree under the internal node will correspond to the SA-range of strings having P as the prefix.

3.10 Forward and Backward Search

The backward search for a string is done in a somewhat unnatural way from right to left. However, sometimes algorithms require patterns to be searched from left to right. Let T^* be constructed by reversing $T[1, n-1]$, and appending a "$" to it. Now, searching for a pattern P in T is equivalent to searching for the reverse of the pattern in T^*. in terms of the BWT terminology, searching for P in T is equivalent to performing a backward search for the reverse of P in T^*. This backward search would actually search characters in P from left to right. This method of searching for a substring is called the forward search, and the index of T^* is called the forward index.

Chapter 4

Survey of Alignment Methods

4.1 Introduction

Sequence alignment is a classical problem, and its applications to biology have been around since late 1960's [72] and with the work of Needleman and Wunsch [113], became well established. As biological data grew at a rapid pace, better algorithms that can deliver results fast on available computing resources were needed. Algorithms like BLAST [7], BLAT [58] and BWT-SW [66] were able to provide efficient alignments of long sequences (usually generated by Sanger sequencing) to large databases. However, with the advent of NGS, it was necessary to align millions of queries to large genomes, and the existing software were not designed for such a task. Furthermore, the NGS sequences carried different error profiles from Sanger sequencing, had additional error information and sometimes used special encoding schemes. Therefore, a new generation of read aligners were born specifically to handle NGS reads.

Character	Name of the operation	Edit operation
m	Match	Reference and the read bases are matched.
r	Mismatch	Replace current base with a different base.
i	Insert	Insert a base to the read.
d	Delete	Delete a base from the read.

Table 4.1: *The list of possible text operations in an edit transcript. A string can be transformed into another string by applying these operations from left to right on the original read.*

4.2 Basic Concepts

4.2.1 Alignments and Mapping Qualities

Given a reference genome T and a read R, the function of an aligner is to return the most likely position(s) in the genome where R might have originated from. The positions that an aligner returns are called hits, mappings or alignments. Generally, a hit or a mapping is considered to be correct if it overlaps the neighbourhood of the origin of the read. However, an alignment is considered to be correct only if the placement of the bases of the read are correctly given. The exact way a read is aligned to a substring T' of T can be described by a so-called edit transcript. An edit transcript is a string on the alphabet $\{m, r, i, d\}$. Each letter in an edit transcript describes a text operation that can be performed on the read, and when these operations are applied from left to right on the read, T' can be produced. Table 4.1 indicates these text operations. A common way to indicate an alignment is to use the CIGAR (Compact Idiosyncratic Gapped Alignment Record) code. The CIGAR code encodes the edit transcript so that the number of times a text operation must be performed is followed by a corresponding CIGAR operator. Operator for matches and replacements are both denoted by `M`. The operators for insertion and deletions are `I` and `D` respectively. A soft-clip operator, denoted by `S`, can appear at the ends of a CIGAR string, and indicates that the corresponding bases should be clipped or ignored.

When determining the goodness of alignments, two scores are generally used. The first score is the Hamming distance which can be used when there are no indels in the alignment. The Hamming distance is the number of mismatches in the read alignment. Note that to calculate the Hamming distance, the read and T' must have the same length, and the transcript should only use the letters m and r. The second score, called the edit distance or the Levenshtein distance, is obtained by assigning values to each character in the edit transcript of the read alignment and summing them up. There are other scoring schemes that adds a penalty whenever a contiguous set of indels are encountered (gap-open penalty) and some scoring schemes do not penalise indels that occur at the beginning and the end of the alignment. Hamming distance calculation can be done in $O(|R|)$ time while finding the best alignment with the other scores need dynamic programming methods that take $O(|R||T|)$ time. Two popular dynamic programming approaches used to determine the optimal alignments are the Smith-Waterman [133] and the Needleman-Wunsch [113] methods.

A good aligner will also return a value, called the mapping quality or the mapQ, for each mapping to indicate the correctness of the mapped location. In cases where the mapping location for a read cannot be resolved unambiguously, the aligner will assign a mapping quality of 0 to the mapping.

4.3 Seeds

In the previous chapter, we have seen efficient methods to search for exact matches. When inexact matches of a string are searched for, a common technique is to use the "seed and extend" method. A seed is a subsequence of the query string. In the seed and extend method, a filtration step identifies all occurrences of a seed in the reference genome. It narrows down the possible locations of inexact matches of the query string. Then, a verification is run on these locations to see if an inexact match for the whole query exists, using a criterion like the edit or the Hamming distance. This step is called

the extension step.

The most basic type of seed is a contiguous seed, which is a q-mer substring of the query string. Classic aligners like BLAST and MegaBlast [156] use this type of seeds, where they search for 11-mer and 28-mer seeds respectively. The dilemma faced when using contiguous seeds is that if the seed is small, the search becomes more sensitive but will return many false locations to verify. On the other hand longer seeds will miss out some correct hits. In [91] it was noted that a compromise could be made using 'spaced seeds', where non-consecutive characters of the query are used as a seed. Such a seed can be specified using a template consisting of 1's and 0's, where 1 stands for a base that will be sampled, and 0 stands for a base that will be ignored. The weight of a spaced seed is the number of 1's in its template. For example the template 010101 is of weight 3, and specifies that the seeds are to be constructed by sampling the characters in even-numbered positions of a string of length 6. When the read length, the sensitivity required and the memory usage are specified, [83] shows how to construct the minimal number of spaced seeds.

Seeds are used in conjunction with an index. In the context of aligners, either the reference genome or the reads can be indexed. It is common practice to use a hash table as the index. Hash tables for large references require a lot of memory when the length of the seed is large. Processing reads in batches and hashing them is less memory intensive, but may become time consuming as it is not a one-time operation like hashing the reference.

4.4 Mismatch Scanning With Seeds

According to the pigeon-hole principle, if a query string R has k-mismatches and it is partitioned in to $(k+1)$ parts, at least one of the partitions should contain an exact match. Using this method, we can design $k+1$ contiguous seeds having a weight of at most $\lceil |R|/(k+1) \rceil$ to detect k-mismatch hits. However, gapped seeds with higher

weights can be designed to handle mismatches by dividing the query into $k+r$ segments, which results in at least r exact matching seeds.

4.5 q-grams

When there are indels in the query string, seeding methods described above will in general fail. q-gram method are more suitable to handle these type of cases. The q-grams are similar to contiguous seeds, but the way they are used is somewhat different. Instead of relying on a single long seed to initiate hit verification, q-gram methods rely on t matching q-grams clustering into a small region to initiate verification. This number t is decided by the q-gram lemma [19, 55] which states that if a query of length w has at most k mismatches and indels, the query and a length w substring in the reference shares at least $t = (w + 1)(k + 1)q$ common q-grams.

4.6 Brief Overview

In this section we will give a brief overview of how sequence aligners evolved. In the beginning, the read lengths produced by the popular NGS technologies were quite small. For example the earliest reads produced by the Illumina and SOLiD sequencers were around 25-36bp in length. The major concern when aligning reads of this length is mismatches, and if reliable indel detection was needed, paired end reads had to be used. Therefore, the very first generation of sequence aligners concentrated on devising efficient methods to map reads allowing mismatches. A seminal publication that set the tone for this generation of mappers is MAQ [77].

The earliest algorithms for finding mismatch hits were seed and extend methods [26, 77, 132]. These algorithms did not scale well with the read length and the number of mismatches. However, it was observed that errors tend to accumulate at the end of the reads so that most of the reads can be aligned reliably by mapping their first 28

bases allowing at most two mismatches [77]. Therefore, a popular approach was to first map this high quality part of the read allowing around two or three mismatches. Once the high quality part of a read was anchored, the rest of it can be aligned around the anchor [69, 74]. Nevertheless, some aligners performed mapping by covering the whole read with seeds [83, 132]. Although mismatch-based seed extension is fast, its not the most accurate in the presence of indels and fails to take full advantage of the encoding scheme of SOLiD. Therefore, some programs used a dynamic programming approach [122, 45] for seed extension.

Along with NGS, computing resources available for processing evolved. Computers having or exceeding 4GB of memory became increasingly common. Taking advantage of this fact, the Bowtie [69] aligner broke new grounds by using the FM-index to index the reference genome. Using clever heuristics, it provided sequence alignments that are accurate, sensitive and upto 35-300 times faster compared to its competitors. This launched the next generation of fast sequence aligners that included BWA [74], SOAP2 [80] and BWA-SW [75]. BWA suite became the de facto standard for accurate aligners and Bowtie became the de facto standard for efficient aligners. Another change that can be observed during this period is that the aligners shifted to indexing the reference rather than the reads.

As longer reads became available, they were more likely to be affected by structural variations, reference mis-assemblies and indels. Therefore, mapping allowing mismatches was not sufficient and the handling of indels became important. Some popular aligners (e.g. [69, 77]) required paired end reads to resolve indels. However, for read lengths of around 75bp, the approach of anchoring the high quality part of the read and extending it allowing variations still produced good results at an acceptable speed [74]. Although some aligners designed to handle shorter reads can be tuned to find indels, it will badly affect the run time. Due to these considerations, the general approach shifted towards using multiple seeds spread throughout the read [68, 75] or q-mers [145] and finding

local matches. In a sense, the strategies have come a full circle to resemble those of classical aligners like BLAST.

A large number of sequence aligners have being developed as shown by Figure 4.1. These aligners make various improvements over seeding strategies, methods of seed extension, ranking of alignments and the types of indexes used. However, their basic methodologies have not changed radically.

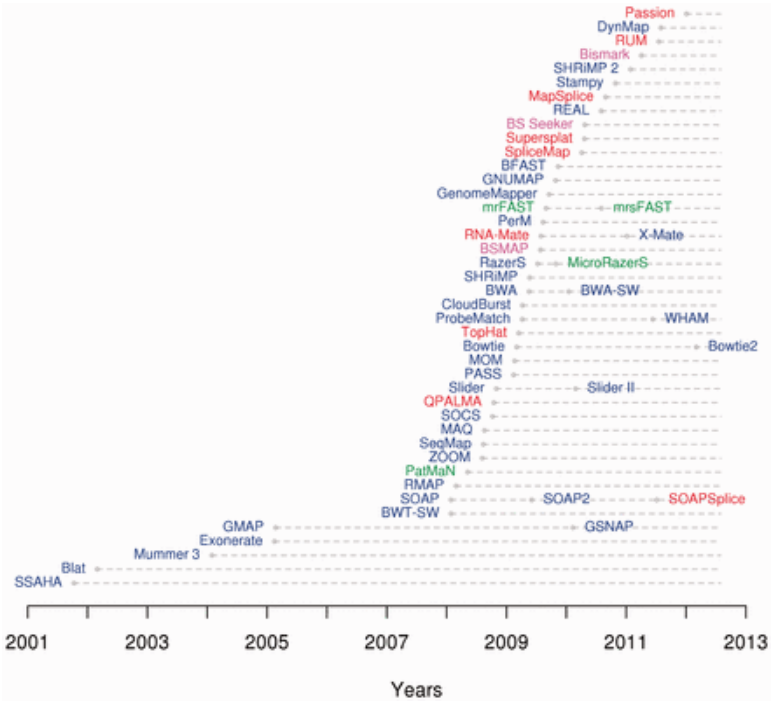


Figure 4.1: *The graph shows the details of a collection of peer-reviewed sequence aligners and their publication date. The graph plots DNA aligners in blue, RNA aligners in red, miRNA aligners in green and bisulphite aligners in purple. An update of a previous version is plotted connected by a grey horizontal line to the original mapper. Reproduced from Fonseca et. al. [38]*

.

4.7 Seed-Based Aligners

The main differences between seed-based aligners are the methods they use to design seeds and hashing tables. Old aligners like Eland [26] and MAQ designed the seeds based on the pigeon-hole principle. These use a $k + 2$ partitioning scheme, with Eland using four partitions and MAQ using around five partitions, limiting their ability to handle mismatches at 2 and 3 mismatches respectively. RMAP uses $k + 1$ templates to find k mismatch mappings. However, due to the small weight of the templates, large values of k will deteriorate its performance. ZOOM uses more sophisticated methods to design a minimum number of spaced seeds according to a given set of specifications.

The memory required by the hash tables can be quite large, especially when the reference is hashed. SOAP [79] using a 12bp seed requires 14GB of memory to index the human genome, which is quite high even by todays standards. A simple strategy to reduce the index size is to skip some bases of the reference when building an index [40, 89, 110], which results in the loss of sensitivity. A clever seed design by PerM [22] uses what is called a single periodic spaced seed. These seeds require only a single index since the seeds are generated by sliding a single template across the read. The resulting index requires only 4.5 bytes per base to index the human genome. BFAST [45] uses a novel two level indexing scheme to index spaced seeds of large weight.

An advantage of indexing the reads is that the amount of memory used can be made flexible. However, this approach might not always be effective when the number of reads mapped is small as it carries the overhead of scanning the genome [74]. Modern aligners usually do not index the reads. One reason is that the processors are now equipped with sufficient memory to carry a genomic index. Another reason is that the increase of the volume of data generated by NGS makes hashing of reads take a similar or a larger amount of memory compared with indexing the genome [48].

Table 4.2 summarizes some of the features of popular seed and q-gram based aligners.

4.8 Suffix Trie Based Methods

In the last two years suffix trie based methods, especially those based on the FM-index, have become increasingly popular. The first uses of suffix trie based methods in sequence alignment can be traced to the programs MUMer [64], OASIS [103] and Vmatch [63]. The first two are based on suffix trees. The last one is based on a data structure called the enhanced suffix array that can simulate any operation on a suffix tree with the same time complexity [1]. The BLAST-like alignment program BWT-SW used an implementation of the FM-index, and practically demonstrated the utility of the FM-index in sequence alignment. Although there are suffix array and enhanced suffix array based aligners developed in recent times [44, 130], it is doubtful whether they can sufficiently outperform the FM-index based methods to justify their large memory footprint.

FM-index can be used not only to find exact matches, but also to find inexact matches. This can be done naïvely by introducing mismatches and indels to the query string and searching for their exact matches. A more refined approach is to traverse the suffix/prefix trie in a depth-first search for the query string while introducing mismatches and indels. However, introduction of mismatches and insertions will make the search space grow exponentially.

To tame this exponential growth, the aligners attempt to prune the search space using various methods. Bowtie uses the pigeon-hole principle and forces mismatches to one half of the read when searching for a single mismatch. BWA can determine some branches in the search trie that will result in excessive number of differences with the query string and prune them. SOAP2 uses seeds obtained by dividing the read into $k + 1$ partitions. The exact matches of the seeds are searched for using a suffix trie traversal, and switching between prefix and suffix trie traversals, k mismatch hits can be enumerated [65]. The switching between the two modes of traversals can be easily done using the bi-directional BWT [65]. Bowtie and BWA uses the forward and backward

FM-indexes for traversing prefix and suffix tries.

Using pruning techniques, two or three mismatch hits can be efficiently enumerated in a read length of around 25-32 bases [69, 74]. In a real life dataset, the high quality region is predominantly error free, and high mismatch hits are rare. Therefore, the high quality region of a read can be anchored fast, and extended. As the read lengths become longer, the approach shifts towards locating seeds with a very low number of mismatches and aligning them to the query using a dynamic programming method. The original BWT-SW algorithm performs sequence alignment by sampling the substrings of the prefix tree of the reference and aligning them against the query using dynamic programming. BWA-SW speeds up this process by aligning the prefix trie of the reference against the prefix DAWG of the query using heuristics. Bowtie2 [68] samples a set of seeds from the query string and maps them against the reference allowing at most one mismatch. These mappings are then prioritized using some heuristics and are extended with a fast SIMD-accelerated dynamic programming method. CUSHAW2 [86] uses maximal exact match regions of the read as seeds. BLASR [21] and GEM [94] choose seeds that do not occur more than a given number of times in the reference. Table 4.3 provides information about some of the suffix trie based methods that are available.

4.9 Aligners and Hardware Improvements

Modern computers have grown quite powerful in the last few years. Processors capable of processing Single-Instruction Multiple-Data (SIMD) instructions have now become commonplace. Sequence aligners have benefited by these developments. The FM-index relies on bit counting, and this can be performed efficiently using SIMD instructions. The implementation of BWT-SW comes with such a routine, and completely eliminates one lookup table needed for the FM-index using SIMD instructions. Many processors contain native bit counting instructions, and we can expect these to be utilised in future.

Another area that benefits from SIMD instructions is the dynamic programming used in the seed extension. SIMD based algorithms have been devised that can speed up the standard Smith-Waterman algorithm several times [118, 34]. These algorithms, along with the Myers bit vector algorithm [111] are used by aligners for performing fast dynamic programming [68, 122].

GPU's are becoming common now, and the amount of memory they have keep on increasing. GPUs carry the potential for enormous amount of parallel processing. GPU's have been used to speed up dynamic programming [87]. [88] managed to implement a complete BWT based alignment pipeline, and since then several GPU based aligners [60, 84, 90] have been introduced.

Aligner	seed based	q-mer based	Index		Notes
			Reference	Read	
Eland [26]	✓			✓	Can be considered as the first specialized short read aigner. Designed to map Illumina reads. Allows two mismatches.
MAQ [77]	✓			✓	First mapper to produce a mapping quality.
ZOOM [83]	✓			✓	Can use custom designed seeds according to the mapping criteria and memory availability. Indels are handled by enumerating possible indel patterns in the seed regions.
Stampy [89]	✓		✓		Uses contiguous seeds of length 15. Hashes every 6^{th} base of the reference. Sensitivity is very high, especially for large indels.
SeqMap [53]	✓			✓	Indels are handled by simulating them in the reads during hashing.
RMAP [132]	✓			✓	Mapping can be done so that low quality bases are always treated as matches. No limit in read length and the number of mismatches.
mrFast [6]	✓		✓		Aimed at returning all hits, and at copy number variation studies.[6]
Razers [145]		✓		✓	Can map reads at a guaranteed sensitivity that user specifies, with lowered sensitivity resulting in significantly higher speeds.
SOAP [79]	✓		✓		A large amount of memory (∼14GB) is required for human-sized genomes.
PerM [22]	✓		✓		Single periodic spaced seeds reduce hash table size.
SHRiMP [22]		✓		✓	Aimed at accurately mapping SOLiD reads.
BFAST [45]	✓		✓		Uses empirically derived optimal seed templates for fixed read and genome sizes. Data structure enables using larger q-mers (e.g. $q = 22$ for long reads).
Hobbes [4]		✓	✓		Fast method when finding multiple hits.[4]
SRmapper [40]	✓		✓		Uses a relatively small amount of memory (∼2.5GB) for hashing a human sized genome.
SeqAlto [110]	✓			✓	Achieves speeds comparable to suffix-trie based methods. Uses an adoptive process to stop searching seeds when a confident alignment is found.

Table 4.2: *Summary of several seed and q-mer based aligners.*

Aligner	Index			Notes
	FM-index	Enhanced SA	Suffix Array	
Bowtie [69]	✓			Conducts a quality-aware, greedy, randomized, depth-first search for alignments.
BWA [74]	✓			Provides a good balance of speed, sensitivity and accuracy.
BWA-SW [75]	✓			Modification of BWT-SW algorithm for short reads. Indexes both reads and genomes.
SOAP2 [80]	✓			Uses bi-directional BWT.
Segemehl [44]		✓		One of the earliest attempts at gapped alignment with suffix tries. Processing chromosome by chromosome reduces memory requirement to 6GB.
Bowtie2 [68]	✓			Indels can be found from a single read in contrast with Bowtie 1. Good balance of sensitivity and speed.
CUSHAW2 [86]	✓			Maximal exact matches of the query are used as seeds. Performance improves for longer reads.
BLASR [21]	✓		✓	Aligner is designed for PacBio reads.
GEM [94]	✓			Extremely fast algorithm. Use filtration methods to limit search space.
Masai [130]	✓	✓	✓	Uses a special kind of seeding called approximate seeds.

Table 4.3: *Summary of several prefix/suffix trie based aligners.*

Chapter 5

Survey of RNA-seq Alignment Methods

5.1 Introduction

Apart from the general difficulties faced when mapping NGS reads, mapping RNA-seq reads presents unique challenges. RNA-seq reads are in general made out of different exonic segments of the genome. These segments can have different sizes and the intronic distances between them can be as small as several bases or as large as hundred thousand bases. To complicate matters further pseudogenes, which are genetic sequences that bear a resemblance to the real genes, can result in false mappings due to their similarity with actual genes. However, most of the splicings occur near highly conserved motifs and the size distribution of exon lengths and intron lengths have been studied for many species. This information can be used to guide the RNA-seq mapping process.

Generic sequence aligners described in the previous chapter serve a limited purpose in the analysis of RNA-seq reads. They can be used to map the reads to a reference cDNA database for quantification purposes [109], or to identify known splice junctions [23, 150]. However, when *de novo* splicing events are to be found, specialised aligners are required

that take into account the nature of RNA-seq reads. In the following sections we will survey the methods used by these specialized RNA-seq aligners.

5.2 Evolution of RNA-seq Mapping

Expressed sequence tag (EST) [2] technology was a predecessor to RNA-seq, and produced relatively long reads that were several hundred bases long. Several methods have been used to align them [58, 131, 153, 70] that take into account the nature of splice sites. However, these methods are not very useful to align the first generation of NGS reads due to their shorter length and larger volume. Earliest studies used known exons to align RNA-seq data [109]. The splice junctions were detected by mapping the reads to a database created by concatenating adjacent exons. If a read maps to one of these, and cannot be mapped to the reference genome, a splice junction can be called.

One of the earliest RNA-seq mappers is QPALMA [14]. QPALMA is a machine learning algorithm, and requires a known set of splice junctions as a training set. Furthermore, it was not a very fast algorithm. Tophat [137] can be considered as the first RNA-seq aligner that was able to do de novo splice junction discovery at a reasonable speed. Tophat creates an approximation of the transcriptome using the coverage of reads that can be mapped to the reference allowing a few mismatches. Then, splice junctions are searched for in the initially unmapped reads using this tentative transcriptome as the guide.

When the read length of NGS reads increased, fast seed-based methods were used to discover splice junctions. For example, a 50bp read can be split into two 25bp seeds, and each part is likely to occur only a small number of times in the reference. These seeds can be rapidly aligned using a fast aligner like Bowtie. Assuming that the seeds come from an exonic region, the junction can be resolved using a seed-and-extend method. However, the extension step is not straightforward. This method of RNA-seq alignment is called split-mapping. Split-mapping has the ability to precisely determine the splice

junctions using a single read.

5.3 Classification of RNA-seq Mappers

5.3.1 Exon-First and Seed-Extend

According to [39], RNA-seq methods can be broadly classified into two classes called exon-first and seed-extend. According to this classification, the exon-first approach maps the whole read into the reference first. We will call this step the exon mapping step. The idea is that most of these reads probably originated completely inside the exons. Those reads that remain unmapped are called the initially unmapped reads (IUM [137]). They are analysed for splice junctions (See Figure 5.1(a)). If the read length is too long, exon mapping may be done by dividing the reads into shorter sub-reads. Aligners in this category include TopHat, MapSplice [142], HMMsplicer [29], SOAPsplice [50], PASSion [155] and TrueSight [81].

The second class of mappers do not try to make an initial distinction between exonic reads and those containing junctions. They will simply divide a read into seeds, map the seeds to the genome and examine them for splice junctions (See Figure 5.1(b)). Aligners in this category include QPALMA, MapNext [10], PALMapper [52], SpliceMap [9], SplitSeek [8], GSNAP [152], Supersplat [18] and SeqSaw [143].

Although many mappers perform an initial exon mapping, how much they depend on the resultant mapping for junction finding varies from aligner to aligner. For example, Splicemap detects junctions completely independent of the exon mapping. However, in Tophat and G-Mo.R-Se [27] it is an integral part of the junction finding process.

Tophat and G-Mo.R-Se use the exon mapping to generate an approximation of the transcriptome, which is called the tentative transcriptome, as follows. Tophat maps reads allowing upto 10 hits. The read islands that are formed by this mapping are

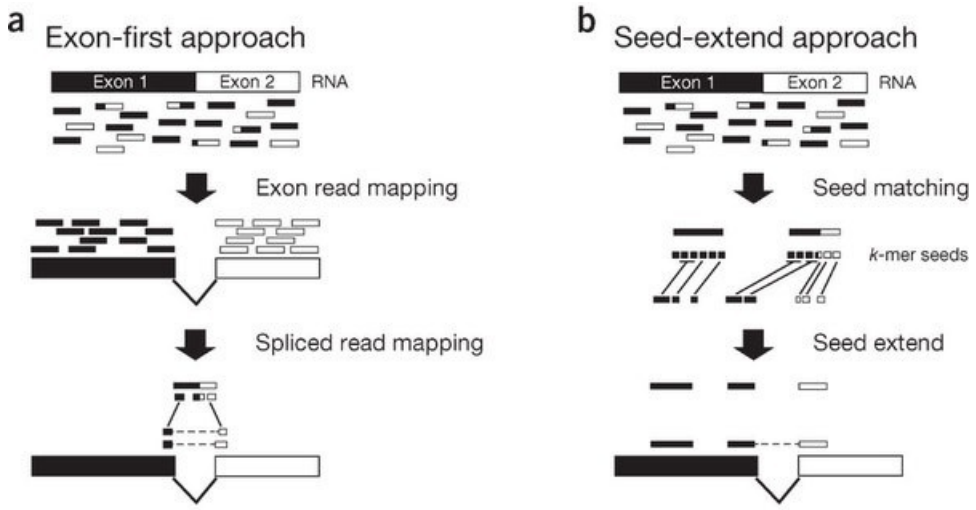


Figure 5.1: *An illustration of exon-first and seed-extend methods. The black and white boxes indicate exonic origins of some reads. (a) Exon-first methods align the full reads to the genome first (exon read mapping), and the remaining reads called IUM reads are aligned next, usually by dividing them into smaller pieces and mapping them to the reference. (b) Seed-extend methods map q-mers of the reads to the reference (seed matching). The seeds are then extended to find splice sites (seed extend). Reproduced from Garber et. al. [39]*

joined together if they are nearby, or extended by attaching about 45bp from their flanking regions. G-Mo.R-Se [27] clusters unique reads and those clusters that have sufficient coverage are retained to create tentative exons called covtigs. The IUM's are then mapped so as to form junctions near the endpoints of the covtigs and islands.

A disadvantage of these two methods is that the read coverage needs to be high in order to get a reliable approximation of the actual transcriptome, and isoforms with a low level of expression may not be detected. However, this method can be used to detect junctions out of reads that retain only a very small portion (like 5bp) from one exon, and is one of the best practical methods to handle very short reads around 36bp.

Exon mapping can speed-up the RNA-seq alignment process as it is a quick way to identify probable junction reads, which can be further investigated using more time consuming methods. However, there is a danger of losing real junction reads by accidentally mapping them at the exonic mapping stage, for example due to the presence of retrotransposed pseudogenes (These pseudogenes are created when the mRNA transcript of a gene is inserted to a chromosome by spontaneous reverse transcription.) Aligners can use an annotation of a known transcriptome [41, 59] to make exon mapping more accurate. For the reference genome of a highly polymorphic species, seed and extend methods give better performance [105].

5.3.2 Annotation-Based Aligners

Genomic annotations of good quality have been produced in recent times for a large number of species. Some methods takes advantage of these annotations to align reads. RNA-Mate [23] and X-Mate [150] map reads to the reference genome and to a library of known junctions. Reads that fail to map are mapped recursively by trimming them at the low-quality ends. SpliceSeq [123] aligns reads to a database of known splicing patterns represented as gene transcript splice graphs. These aligner might be useful when de novo discovery of splice junctions is not needed, for example when the expression of

known genes are being studied. In fact, many aligners support some form of annotation usage. These aligners might use the annotation to make the junction predictions more accurate during or after performing de novo junction finding [9, 137, 10], or they might map the reads to the annotation before de novo junction discovery [59, 41].

5.3.3 Learning-Based Approaches

One of the earliest aligners, QPALMA, uses support vector machines to find junctions. However it cannot perform true de novo junction discovery since it requires an initial set of junctions as a training set. However, the hidden Markov model based HMMsplicer creates its own training data set. It does so by splitting the IUM reads into halves, mapping them to the reference and retaining the aligned halves as the training set. PALMapper [52] is a version of QPALMA that has been sped up by replacing its core aligner with a new aligner called GenomeMapper [128].

Most RNA-seq aligners assume a mammalian gene model of the reference, and are biased towards canonical and semi-canonical junctions. However, learning-based methods can perform unbiased splice junction discovery and might be more suitable for species whose gene models are not known. A summary of some of RNA-seq mappers is given in Table 5.1

5.4 Splice Junction Finding

To determine a splice junction, we need a read that passes over the junction. The first step to identifying junctions from such a read involves determining a segment each from the distinct exons that constitute the junction. Then the precise break point can be determined by extending them.

There are several ways of determining the exonic parts of the junctions. One way is to partition the read into q-mers and to find two q-mers that are not too close to form a contiguous block, perhaps allowing some margin for small indels. Then a junction

Aligner	Exon mapping	Seed Extend	Splice Model		Notes
			Unbiased	Biased	
QPALMA [14]		✓			Uses support vector machines for splice prediction.
G-Mo.R-Se [27]	✓		✓		Method is somewhat similar to Tophat.
HMMSplicer [29]	✓				Uses a hidden Markov model of splice junctions.
Tophat [137]	✓			✓	Depends on building a tentative transcriptome.
Tophat2 [59]	✓			✓	Does not totally rely on tentative transcriptome. Can handle indels.
SpliceMap [9]		✓		✓	Uses one half of the read as a primary anchor, and a 10bp residue as secondary anchor.
MapSplice [142]	✓		✓		One of the most sensitive mappers.
MapNext [10]		✓		✓	Very high memory usage.
SOAPSplice [50]	✓			✓	Can align to junctions reads having at least 8bp from one exon.
SpliceSeq [123]					Dependant on a genomic annotation.
SplitSeek [8]		✓	✓		Designed for SOLiD reads.
Supersplat [18]		✓	✓		Very high memory usage (around $\sim$ 14GB).
RUM [41]	✓				Uses BLAT to align IUM reads.
PASSion [155]	✓			✓	Can align only paired reads.
GSNAP [152]		✓		✓	Very sensitive method.
OLEgo [151]	✓			✓	Targeted at finding small exons.
STAR [30]		✓	✓	✓	Very fast aligner. Uses longest exact matches as seeds.
SeqSaw [143]		✓	✓		Memory requirement can be high.
TrueSight [81]	✓	✓		✓	Uses a machine learning approach to junction finding.
PALMapper [52]		✓		✓	Modification of QPALMA.
RNA-Mate [23]					Maps using an annotation.
X-Mate [150]					Enhanced version of RNA-Mate.

Table 5.1: *Summary of several RNA-seq aligners. Splice model denotes the approach taken to resolve a junction. Biased methods prefer or only consider known junction signals, while unbiased methods do not prefer any known junction signal type.*

search between these two q-mers can be initiated [142, 10, 8, 151]. Another method anchors one part of the read to the genome. Next, a substring from the un-anchored part of the read is taken, and the neighbourhood of the primary anchor is searched for its occurrences. If the substring can be found, using it as the second anchor, a junction search can be initiated [9, 142, 151, 50].

A third method to identify junctions involves searching IUM reads for splice junctions deduced from a tentative transcriptome [137, 27]. To facilitate this, a hash table is built to query occurrences of q-mers in the IUM reads. Examining the edges of exons in the tentative transcriptome for splice motifs, a possible list of donor and acceptor sites can be drawn up. From this list, by pairing suitable acceptors and donors, a set of tentative junctions can be found. Then, q-mers near these tentative junctions are taken, and reads containing these q-mers are identified using the hash table. Finally, junctions are validated by aligning these reads onto them.

Determining the exact breakpoint of a splice junction can be done by searching for junction motifs during the extension. Many aligners search for only conservative and semi-conservative junction signals. If an unbiased junction search is performed, factors like the number of mismatches in the alignment [9] and read coverage [8] can be used. However, the junctions that do not contain canonical splicing signals usually tend to be false junctions.

Chapter 6

k-Mismatch Alignment Problem

6.1 Introduction

Due to the differences between the sampled genome and the reference genome, and the errors introduced during the sequencing, NGS reads need to be mapped allowing a reasonable number of errors. Mapping NGS reads in general require the ability to handle indels. However, for platforms like Illumina and SOLiD, most of the reads can be aligned allowing mismatches only. In fact, some popular aligners like Bowtie [69] only consider mismatches in alignment, while many others consider only mismatches by default [83, 145]. Also, there are experiments where a large number of mismatches are allowed, sometimes along with indels [32, 97]. Therefore, the k-mismatch problem, i.e. mapping a short read allowing k-mismatches to a reference genome is an interesting problem in bioinformatics.

Although sequence aligners can handle a small number of mismatches efficiently, the running times will increase rapidly when the number of mismatches increases. For hash-based aligners, the number of seeds required will increase with the number of mismatches, and the mapping slows down due to the increase in hash table lookups and the seed extensions. BWT-based aligners become slow due to the rapid increase of

branches that needs to be traversed as the number of mismatches increases. As shown in Section 6.6, aligners can be slow or inadequate to handle even moderate numbers of mismatches. To overcome the slowdown with large mismatches, aligners use various heuristic methods. These methods can cut down the alignment time dramatically. However, applying these heuristics to solve the k-mismatch problem will result in a loss of sensitivity and accuracy.

Different types of experiments require different types of mappings. The most basic type of alignment reports the first hit of a read satisfying a given mismatch threshold. However, in many experiments, hits are required to satisfy some form of a uniqueness criterion. Other situations might require reporting multiple hits for each read. For example, in ChIP-seq experiments, scientists might prefer to map reads uniquely for better accuracy or they might allow multiple mapping for better sensitivity. Many RNA-seq pipelines need an external aligner to produce mappings of a given read, which are then post-processed for splice junctions. These pipelines usually require multiple mappings of a given read [9, 142, 137], since the first or the unique hits may map the read to pseudo-genes or map a read covering a splice junctions to a contiguous region in the genome. Although an aligner can be designed to perform extremely well for a first hit search, it might perform relatively slow for multiple and unique mappings. Therefore, it is preferable to have a mapping algorithm that can efficiently handle all these common requirements.

I introduce a new exact method BatMis (*B*asic *A*lignment *T*ool for *Mis*matches) that solves the k-mismatch problem much faster than existing methods. BatMis does not use heuristics. It is an exact method which aligns a read to the reference genome with the minimum number of mismatches and can align a read allowing up to 10 mismatches in the whole read. Our benchmarks show that BatMis is at least as fast as the current aligners. It also shows that in many cases, BatMis can compete favorably even when compared with the heuristic modes of other aligners. The material in this Chapter is

based on my publication Tennakoon et al [135].

6.2 Problem Definition

We will now define the k-mismatch mapping problem. Let X and Y be two strings of equal length. Denote the Hamming distance between X and Y by $d(X, Y)$. Consider a genome T and a string R. The k-mismatch problem is to find all positions i such that $d(R[1, |R|], T[i, i+|R|-1]) \leq k$. We are also interested in reporting the occurrences in the order of increasing mismatches, i.e., we report i before i' if $d(R[1, |R|], T[i, i+|R|-1]) < d(R[1, |R|], T[i', i'+|R|-1])$.

6.3 Description of the Algorithm

Consider a reference T. Let R be a read and K be a mismatch threshold. Solving the k-mismatch problem is to find the set of all strings x in T such that $d(x, R) \leq K$. Let H_R^k be the set of all substrings x in T such that $d(x, R) = k$ for $k = 1, \ldots, K$. Then, our aim is equivalent to computing $\bigcup_{k=0}^{K} H_R^k$.

We define $R_l = R[1, \lfloor |R|/2 \rfloor]$ and $R_r = R[\lfloor |R|/2 \rfloor + 1, |R|]$ to be the left and right halves of R respectively. Our proposed algorithm BatMis is a seed-and-extend method. It has two phases. Phase 1 finds all substrings in T which look similar to R_l and R_r by recursion. Precisely, it computes the set of seeds $H_{R_l}^k$ and $H_{R_r}^k$ for a set of values of k not exceeding $\lfloor K/2 \rfloor$. In Phase 2, the patterns found in Phase 1 are extended to get all k-mismatch patterns of R.

6.3.1 Seeding

Lemma 6.3.1 provides us with a minimal set of H_l^k and H_r^k guaranteed to find all k-mismatch patterns of R with this algorithm.

Lemma 6.3.1. *Consider a read R and a string R' of equal length such that $d(R, R') \leq k$, where $k \geq 1$. We have two cases.*

- **Case 1: k is even.** *We have either $d(R'_l, R_l) \leq k/2$ or $d(R'_r, R_r) \leq k/2 - 1$.*
- **Case 2: k is odd.** *We have either $d(R'_l, R_l) \leq (k-1)/2$ or $d(R'_r, R_r) \leq (k-1)/2$.*

Proof. When k is even, from the pigeon hole principle, we have the cases $d(R'_l, R_l) \leq k/2$ or $d(R'_l, R_l) > k/2$. In the first case, the proof is obvious. In the second case we have

$$d(R'_r, R_r) \leq k - d(R'_l, R_l) \leq k - k/2 - 1 = k/2 - 1.$$

Hence case 1 follows. Similarly, we can show that case 2 is true. □

We can now re-state the algorithm as follows. When k is even, we extend patterns in $\bigcup_{i=0}^{k/2} H^i_{R_l} \cup \bigcup_{i=0}^{k/2-1} H^i_{R_r}$ to obtain all k-mismatch patterns of R. When k is odd, we extend patterns in $\bigcup_{i=0}^{(k-1)/2} H^i_{R_l} \cup \bigcup_{i=0}^{(k-1)/2} H^i_{R_r}$ to obtain all k-mismatch patterns of R.

6.3.2 Extension

The phase 2 of the algorithm where the extension of patterns are performed, is done using the procedures $PExt$ and $SExt$. Given a set X of substrings of T, $PExt(X, R, k)$ performs prefix extension of the strings x in X to form another set Y of strings $y = x \cdot \alpha$ of T until every string $y \in Y$ satisfies either (1) $|y| = |R|$ and $d(y, R) \leq k$ or (2) $d(y, R[1, |y|]) = k + 1$. Similarly, $SExt(X, R, k)$ performs suffix extension of the strings x in X to form another set Y of strings $y = \alpha \cdot x$ of T until every $y \in Y$ satisfies either (1) $|y| = |R|$ and $d(y, R) \leq k$ or (2) $d(y, R[|R| - |y| + 1, |R|]) = k + 1$. The procedures use the following recurrences and their pseudocodes are shown in Algorithms 6.1-6.2.

- If $|X| > 1$, $PExt(X, R, k) = \bigcup_{x \in X} PExt(\{x\}, R, k)$

- If $X = \{x\}$, $PExt(\{x\}, R, k) =$

$$\begin{cases} \{x\}, \text{ if } |x| = |R| \text{ or } d(x, R[1, |x|]) = k + 1 \\ \bigcup_{\substack{\sigma \in \Sigma \\ x \cdot \sigma \in T}} PExt(\{x \cdot \sigma\}, R, k), \text{ otherwise} \end{cases}$$

- If $|X| > 1$, $SExt(X, R, k) = \bigcup_{x \in X} SExt(\{x\}, R, k)$

- If $X = \{x\}$, $SExt(\{x\}, R, k) =$

$$\begin{cases} \{x\}, \text{ if } |x| = |R| \text{ or } d(x, R[|R| - |x| + 1, |R|]) = k + 1 \\ \bigcup_{\substack{\sigma \in \Sigma, \\ \sigma \cdot x \in T}} SExt(\{\sigma \cdot x\}, R, k), \text{ otherwise} \end{cases}$$

Algorithm 6.1: PExt(X, R, k)

Data: A set X of substrings in T of length at most $|R|$ and have less than k mismatches with a prefix of R.
Result: A set $\{z \mid x \in X, z = x \cdot y \in T, |z| \leq |R|, d(z, R[1, |z|]) = k + 1\} \cup \{z \mid x \in X, z = x \cdot y \in T, |z| = |R|, d(z, R) \leq k\}$.

```
Set S = ∅;
if |X| > 1 then
    for each x ∈ X do
        S = S ∪ PExt({x}, R, k);
    end
else
    Let x be the string in X;
    if d(x, R[1, |x|]) ≤ k and |x| < |R| then
        for x · σ ∈ T where σ ∈ {a, c, g, t} do
            S = S ∪ PExt({x · σ}, R, k);
        end
    else
        S = S ∪ {x};
    end
end
return S
```

Algorithm 6.2: SExt(X, R, k)

Data: A set X of substrings in T of length at most $|R|$ and have less than k mismatches with a suffix of R.

Result: A set
$\{z \mid x \in X, z = y \cdot x \in T, |z| \leq |R|, d(z, R[|R| - |z| + 1, |R|]) = k + 1\} \cup$
$\{z \mid x \in X, z = y \cdot x \in T, |z| = |R|, d(z, R) \leq k\}$.

```
1  Set S = ∅;
2  if |X| > 1 then
3      for each x ∈ X do
4          S = S ∪ SExt({x}, R, k);
5      end
6  else
7      Let x be the string in X;
8      if d(x, R[|R| − |x| + 1, |R|]) ≤ k and |x| < |R| then
9          for σ · x ∈ T where σ ∈ {a, c, g, t} do
10             S = S ∪ SExt({σ · x}, R, k);
11         end
12     else
13         S = S ∪ {x};
14     end
15 end
16 return S
```

6.3.3 Increasing Efficiency

Given these two procedures, the k-mismatch patterns of R can be computed as follows. When k is even, we report $\{x \in PExt(\bigcup_{i=0}^{k/2} H_{R_l}^i, R, k) \cup SExt(\bigcup_{i=0}^{k/2-1} H_{R_r}^i, R, k) \mid |x| = |R|, d(x, R) = k\}$. When k is odd, we report $\{x \in PExt(\bigcup_{i=0}^{(k-1)/2} H_{R_l}^i, R, k) \cup SExt(\bigcup_{i=0}^{(k-1)/2} H_{R_r}^i, R, k) \mid |x| = |R|, d(x, R) = k\}$.

The above procedure not only computes all k-mismatch patterns of R, but also reports them in the increasing order of the number of mismatches. However, it is slow since it performs a lot of redundant computations. We can modify the seed extension routine to avoid redundant computations. We divide our seed extension procedure into $K + 1$ iterations. For the k^{th} iteration where $k = 0, 1, \ldots, K$, our procedure tries to obtain H_R^k, i.e., all k-mismatch patterns of R. In the 0^{th} iteration, we set $H_R^0 = \{R\}$ if R exists in T; and set $H_R^0 = \emptyset$ otherwise. For the remaining iterations, we will not generate H_R^k starting from the scratch. Instead, our routine will check all the unsuccessfully extended patterns from the $(k-1)^{\text{th}}$ iteration and see if they can be extended and become a k-mismatch pattern of R. Precisely, the k^{th} iteration is divided into two stages. The first stage tries to extend those unsuccessfully extended patterns from the $(k-1)^{\text{th}}$ iteration. The second stage tries to recover the remaining k-mismatch patterns by extending a special set of seeds that guarantees to generate all the remaining k-mismatch patterns with no redundancy.

6.3.4 Utilizing Failed Extensions

Before we give the details of Phase 2, we need some definitions to describe the set of unsuccessfully extended patterns from the $(k-1)^{st}$ iteration. By Lemma 6.3.1, if $k-1$ is odd, we need to extend the patterns in $\bigcup_{i=0}^{(k-2)/2} H_{R_l}^i \cup \bigcup_{i=0}^{(k-2)/2} H_{R_r}^i$ to obtain all the $(k-1)$-mismatch patterns of R. If $k-1$ is even, we need to extend the

patterns in $\bigcup_{i=0}^{(k-1)/2} H^i_{R_l} \cup \bigcup_{i=0}^{(k-3)/2} H^i_{R_r}$ to obtain all the $(k-1)$-mismatch patterns of R. When the extended patterns accumulate k mismatches, their extensions are stopped and are marked as unsuccessfully extended patterns. These unsuccessfully extended patterns are included in PRE^k_R and SUF^k_R depending on whether they have k mismatches with a prefix or a suffix of R respectively. Formally, they are defined as follows. (Note that $\lceil k/2 \rceil - 1 = (k-2)/2$ if $k-1$ is odd and $(k-1)/2$ if $k-1$ is even. Also, $\lfloor k/2 \rfloor - 1 = (k-2)/2$ if $k-1$ is odd and $(k-3)/2$ if $k-1$ is even.)

- Let PRE^k_R be the set of substrings x in T such that $d(x, R[1, |x|]) = k$, $x[|x|] \neq R[|x|]$ and $d(x[1, |R_l|], R_l) \leq \lceil k/2 \rceil - 1$.
- Let SUF^k_R be the set of substrings x in T such that $d(x, R[|R| - |x| + 1, |R|]) = k$, $x[1] \neq R[|R| - |x| + 1]$ and $d(x[|x| - |R_r| + 1, |x|], R_r) \leq \lfloor k/2 \rfloor - 1$.

Intuitively, PRE^k_R contains a subset of the shortest substrings of T having exactly k mismatches with a prefix of R, and SUF^k_R contains a subset of shortest substrings of T having exactly k mismatches with a suffix of R. The following lemma states how to compute the sets H^1_R, PRE^2_R, and SUF^2_R.

Lemma 6.3.2. *Consider a read R. Let $P = PExt(H^0_{R_l}, R, 1)$ and $S = SExt(H^0_{R_r} \cup SUF^1_R, R, 1)$.*

a) $H^1_R = \{x \in P \cup S \mid d(x, R) = 1\}$.

b) $PRE^2_R = \{x \in P \mid d(x, R[1, |x|]) = 2\}$.

c) $SUF^2_R = \{x \in S \mid d(x, R[|R| - |x| + 1, |R|]) = 2\}$.

Proof. By Lemma 6.3.1, for any string R' which has 1 mismatch with R, we have either $R'_l = R_l$ or $R'_r = R_r$. Hence, H^1_R contains all strings in $P \cup S$ whose patterns have exactly 1 mismatch with R. The equations for PRE^2_R and SUF^2_R follow by definition. □

6.4 The BatMis Algorithm

The following two lemmas state the recursive formulas to compute H_R^k, PRE_R^{k+1} and SUF_R^{k+1} for $k \geq 2$.

Lemma 6.4.1. *Consider a read R and suppose k is odd. Let $P = PExt(PRE_R^k, R, k)$ and $S = SExt(H_{R_r}^{\lfloor k/2 \rfloor} \cup SUF_R^k, R, k)$.*

a) $H_R^k = \{x \in P \cup S \mid d(x, R) = k\}$.

b) $PRE_R^{k+1} = \{x \in P \mid d(x, R[1, |x|]) = k + 1\}$.

c) $SUF_R^{k+1} = \{x \in S \mid d(x, R[|R| - |x| + 1, R|]) = k + 1\}$.

Proof. Let R' be any string in T such that $d(R', R) = k$, where k is odd. By Lemma 6.3.1, we have either (1) $d(R'_l, R_l) \leq (k-1)/2 = \lceil k/2 \rceil - 1$ (i.e. $R'_l \in \bigcup_{i=0}^{\lceil k/2 \rceil - 1} H_{R_l}^i$) or (2) $d(R'_r, R_r) \leq (k-1)/2 = \lfloor k/2 \rfloor$ (i.e. $R'_r \in \bigcup_{i=0}^{\lfloor k/2 \rfloor} H_{R_r}^i$). If R' satisfies (1), there should be some λ, where $|R'_l| \leq \lambda \leq |R'|$, such that $d(R'[1, \lambda], R[1, \lambda]) = k$, with $R'[\lambda] \neq R[\lambda]$. By definition, $R' \in \{x \in P \mid d(x, R) = k\}$; if R' satisfies (2), $R' \in \{x \in S \mid d(x, R) = k\}$. Hence, $H_R^k \subseteq \{x \in P \cup S \mid d(x, R) = k\}$. Since H_R^k contains all the k-mismatch strings of R in T, we have $H_R^k \supseteq \{x \in P \cup S \mid d(x, R) = k\}$. From the last two relations the first identity can be obtained.
By definition,

$$\begin{aligned} PRE_R^{k+1} &= \{x \in PExt(\cup_{i=0}^{\lceil (k+1)/2 \rceil - 1} H_{R_l}^i, R, k) \mid d(x, R[1, |x|]) = k + 1\} \\ &= \{x \in PExt(\cup_{i=0}^{\lceil k/2 \rceil - 1} H_{R_l}^i, R, k) \mid d(x, R[1, |x|]) = k + 1\} \\ &= \{x \in P \mid d(x, R[1, |x|]) = k + 1\} \end{aligned}$$

Using similar arguments, we can prove the third statement. □

Lemma 6.4.2. *Consider a read R and suppose k is even. Let $P = PExt(H_{R_l}^{\lfloor k/2 \rfloor} \cup PRE_R^k, R, k)$ and $S = SExt(SUF_R^k, R, k)$.*

a) $H_R^k = \{x \in P \cup S \mid d(x, R) = k\}$.

b) $PRE_R^{k+1} = \{x \in P \mid d(x, R[1, |x|]) = k + 1\}$.

c) $SUF_R^{k+1} = \{x \in S \mid d(x, R[|R| - |x| + 1, |R|]) = k + 1\}$.

Proof. The proof is similar to that of Lemma 6.4.1. □

Algorithm 6.3: BatMis(R, K)

Data: Read R and a mismatch threshold K
Result: Report H_R^k for $k = 0, 1, \ldots, K$

/* **Phase 1: Compute** $H_{R_l}^k, H_{R_r}^k$ **for** $k = 0, 1, \ldots, \lfloor K/2 \rfloor$ */
Compute $\{H_{R_l}^k \mid k = 0, 1, \ldots, \lfloor K/2 \rfloor\}$ by calling $BatMis(R_l, \lfloor K/2 \rfloor)$;
Compute $\{H_{R_r}^k \mid k = 0, 1, \ldots, \lfloor (K-1)/2 \rfloor\}$ by calling $BatMis(R_r, \lfloor (K-1)/2 \rfloor)$;
/* **Phase 2: Compute** H_R^k **for** $k = 0, \ldots, K$ */
Set $H_R^0 = \{R\}$ if R in T and $\emptyset$ otherwise;
for $k = 1$ *to* K **do**
    if $k = 1$ **then**
        $P = PExt(H_{R_l}^0, R, 1)$;
        $S = SExt(H_{R_r}^0, R, 1)$;
    else if k *is odd* **then**
        $P = PExt(PRE_R^k, R, k)$;
        $S = SExt(H_{R_r}^{\lfloor k/2 \rfloor} \cup SUF_R^k, R, k)$;
        $(H_R^k, PRE_R^{k+1}, SUF_R^{k+1}) = Extend(PRE_R^k, H_{R_r}^{\lfloor k/2 \rfloor} \cup SUF_R^k, R, k)$;
    else
        $P = PExt(H_{R_l}^{\lfloor k/2 \rfloor} \cup PRE_R^k, R, k)$;
        $S = SExt(SUF_R^k, R, k)$;
        $(H_R^k, PRE_R^{k+1}, SUF_R^{k+1}) = Extend(H_{R_l}^{\lfloor k/2 \rfloor} \cup PRE_R^k, SUF_R^k, R, k)$;
    end
    $H_R^k = \{x \in P \cup S \mid d(x, R) = k, |x| = |R|\}$;
    $PRE_R^{k+1} = \{x \in P \mid d(x, R[1, |x|]) = k + 1\}$;
    $SUF_R^{k+1} = \{x \in S \mid d(x, R[|R| - |x| + 1, |R|]) = k + 1\}$;
end
return $(H_R^0, H_R^1, \ldots, H_R^K)$

Algorithm 6.3 gives the final BatMis algorithm $BatMis(R, K)$, which computes all k-mismatch patterns of a read R for $0 \leq k \leq K$. It uses Lemmas 6.3.2-6.4.2. The

first phase (lines $1-2$) divides R into two equal halves R_l and R_r and it recursively calls $BatMis(R_l, \lfloor K/2 \rfloor)$ and $BatMis(R_r, \lfloor (K-1)/2 \rfloor)$ to compute $H^k_{R_l}$ and $H^k_{R_r}$ for $0 \leq k \leq \lfloor K/2 \rfloor$ and $0 \leq k \leq \lfloor (K-1)/2 \rfloor$ respectively. The second phase iteratively computes H^k_R for $k = 0, \ldots, K$. It first computes H^0_R (line 3), H^1_R, PRE^2_R, and PRE^2_R by Lemma 6.3.2 (lines $5-7$). Then, the program applies Lemmas 6.4.1 and 6.4.2 to compute $(H^k_R, PRE^k_R, SUF^k_R)$ iteratively for $k = 2, \ldots, K$ (see lines $8-17$). Finally, the program reports H^k_R for $k = 0, \ldots, K$.

6.5 Implementation of BatMis

In order to execute $BatMis$, we need to find all the occurrences of strings corresponding to $H^i_{R_l}, H^i_{R_r}, PRE^j_R$, and SUF^j_R for $0 \leq i \leq \lfloor K/2 \rfloor, 0 \leq j \leq K$. Furthermore, we need an efficient way to extend them. To solve this problem, we build two FM indexes FM_T and FM_{T^*} corresponding to T and its reverse T^* (see Section 3.10) . All strings in $H^i_{R_l}$ and PRE^j_R are represented as SA_{T^*}-ranges, while all strings in $H^i_{R_r}$ and SUF^j_R are represented by their SA_T-ranges, where $0 \leq i \leq \lfloor K/2 \rfloor, 0 \leq j \leq K$. The extension of SA_{T^*}-ranges are done using forward searches. Similarly, the extension of SA_T-ranges are done using backward searches. The strings in H^k_R will in general contain a mixture of SA_T and SA_{T^*}- ranges, and it might be necessary to convert SA_T ranges to SA_{T^*} ranges or vice versa. This conversion is done by taking the string corresponding to SA_T (or SA_{T^*}) range and performing a forward (or backward) search on it. To speedup pattern searching, we build a table containing the SA-ranges of all substrings of length at most 6. This table can be used to calculate the SA-ranges corresponding to the first few bases of a string quickly.

$BatMis$ concatenates individual chromosomes of a genome into a single genome. The exact algorithm is run on this single genome. It might happen that a read will align to a chromosome boundary. These boundary errors occur very rarely. For example, in

the datasets I tested in Section 6.6 at most two such errors occurred per dataset. This accuracy is quite sufficient for practical purposes, but I have included a post-processing script to thoroughly resolve reads with boundary errors.

The algorithm is implemented in C/C++. We use the FM-index routines from BWT-SW [66]. For decoding SA_T and SA_{T^*} ranges, SA_T and $SA_T{}^*$ are sampled at every 8^{th} position. Since storing the sampled SA-ranges take a lot of memory, the program can optionally convert all SA_T ranges to SA_{T^*} ranges (or vice versa) and use only one sampling. For the human genome, if only one sampling is used, the decoding algorithm can be run under 4GB of RAM.

Furthermore, the recursions are unrolled for efficiency. For mismatch thresholds less than 5, the algorithm is implemented as stated. When scans are performed allowing a large number of mismatches, storing SUF_R^i and PRE_R^i require a lot of memory. To reduce the memory usage, for $k > 5$, the set of k-mismatch hits, H_R^k, is computed directly based on Lemma 6.3.1. Although this approach reduces the memory required to store SUF_R^i and PRE_R^i for $i > 5$, it will also generate duplicate hits. Post-processing steps are performed to remove these duplicate hits.

6.6 Results

There are a vast number of sequence aligners that can perform exact k-mismatch alignment. Different aligners have different policies regarding how to rank the hits and how to handle uncalled bases (e.g. N's in the read), which makes straightforward comparison difficult. To allow us to compare different aligners fairly, we use data sets without uncalled bases. We test each program's ability to report the least mismatch hits, the unique hits and multiple hits. By the least mismatch hit of a read R in the reference T, we mean any position i such that $d(R, T[i, i + |R| - 1]) \leq k$, and for any other position j we have $d(R, T[j, j + |R| - 1]) \geq d(R, T[i, i + |R| - 1])$. By a unique hit of a read R in the reference T, we mean a unique lowest mismatch hit of R in T.

For comparison with our algorithm, we chose BWA (version 0.5.9-r16) to represent aligners of the BWT family, RazerS2 to represent q-gram methods and ZOOM (Linux64 demo version 1.5.5.20120225072719) to represent methods that use gapped seeds. These programs are among the best in the literature for handling a large number of mismatches. We did not use the popular aligner Bowtie for our tests since its design is based on using heuristics for handling high mismatches and it does not have true k-mismatch handling capabilities for $k > 3$. In the version 2 of Bowtie, we can only set the number of mismatches in a seed region to 0 or 1. Therefore, it cannot be used in the experiments given here. All the experiments were done on a Linux server running on 2 x 6-Core Intel Xeon X5680 Processors (3.33Ghz), with 144GB RAM. Each aligner was run in a single core, and the user time was reported.

All tested programs were run in their default settings with appropriate options set for ignoring indels and enabling exact or heuristic mapping as needed. The demo version of ZOOM is only fully sensitive up to four mismatches. Because of this limitation, ZOOM was run allowing up to four mismatches only. Both BatMis and RazerS2 can report least mismatch hits and unique hits. For BWA, the default setting will report a unique hit if it exists or will otherwise report a random least mismatch hit. For ZOOM, the default mode outputs only the unique hits; it was made to output least mismatch hits with the -mk option. BWA, ZOOM and BatMis used the same reference genome where uncalled bases in the genome were replaced with a random nucleotide. For BWA and BatMis, the total time for the alignment and decoding the output is reported. For BatMis, post processing was done to recover hits with boundary errors, and this time was added to the total time reported.

6.6.1 Ability to Detect Mismatches.

In this section we examine the robustness of mismatch mappings of the selected aligners. We randomly extracted two sets of 51bp and 100bp reads from hg18. Each

dataset contained 100,000 reads. New k-mismatch datasets were created by introducing exactly k mismatches uniformly at random to all reads in the original dataset for $k = 0, 1, 2, 3, 4, 5, 8, 10$. If an aligner performs k-mismatch mapping correctly, it must be able to map all the k-mismatch reads to the reference genome allowing k-mismatches.

Table A.3 in the Appendix summarizes the results. BWA missed some hits with a large number of mismatches for 100bp reads. It was able to map all the reads with up to five mismatches, but was only able to map 97291 and 15124 of reads having 8 and 10 mismatches respectively. Other aligners were able to map back all the reads. This result suggests that BatMis, ZOOM and Razers2 can detect k-mismatches effectively but BWA might miss some hits when k is large.

6.6.2 Mapping Real Data

This section studies the performance of different algorithms using real data. We first check the performance of each aligner when reporting the least mismatch hits. Many biologists prefer to have unique hits as a criterion to filter out noise. Therefore, we also measure the performance of different aligners on finding unique hits in real data.

The evaluation uses the Illumina sequencing datasets ERR000577 and ERR024201 taken from the European Nucleotide Archive. The datasets contained reads of lengths 51bp and 100bp respectively. These datasets are paired end reads, and we chose the datasets containing the forward reads. Reads containing uncalled bases were filtered out, and the first 1,000,000 reads were selected for benchmarking. These sets of reads were mapped to the reference genome hg18 allowing different numbers of mismatches. The results of these mappings are given in Tables 6.1 and 6.2. ZOOM and Razers2 produce a small number of false mappings and BWA misses some hits. However, BatMis produces accurate mappings. For the 51bp dataset, the number of missing or invalid hits reported is negligible. However, BWA mappings become very unreliable as the number of mismatches are increased. Overall, BatMis is the only aligner that is fully

	51 bp				**100bp**			
	2 mis	3 mis	4 mis	5 mis	4 mis	5 mis	8 mis	10 mis
BatMis	905121	940232	960246	972729	928493	935087	951483	960707
BWA	905121	940232	960246	972726	928493	935087	942562	930632
ZOOM	905121	**940233**	**960247**	-	928493	-	-	-
RazerS2	905121	940232	960246	972729	928493	935087	951483	960707

Table 6.1: *Table showing the least mismatch mappings reported by aligners allowing different numbers of mismatches. 1 000 000 reads from the datasets ERR000577 (51bp) and ERR024201 (100bp) were mapped. All mappers produce the same number of hits upto 2 mismatches and 5 mismatches for 51bp and 100bp reads respectively. BatMis and RazerS2 consistently performs well across all mismatches. However, other aligners report false mappings or under reports hits at high mismatches. The extra mappings in the bold entries for ZOOM are due to incorrect mappings.*

accurate across all read lengths and mismatch thresholds.

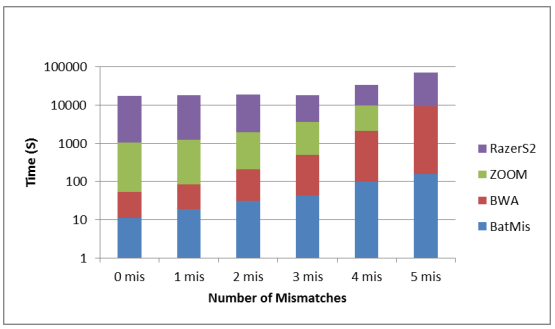
Figure 6.1 shows the comparison of the time taken to report the least mismatch and unique hits. The time is shown scaled logarithmically since the mapping time of BatMis can sometimes be several orders of magnitudes faster than the competitors. For all the read lengths and mismatch settings tested, BatMis is the fastest.

6.6.3 Multiple Mappings

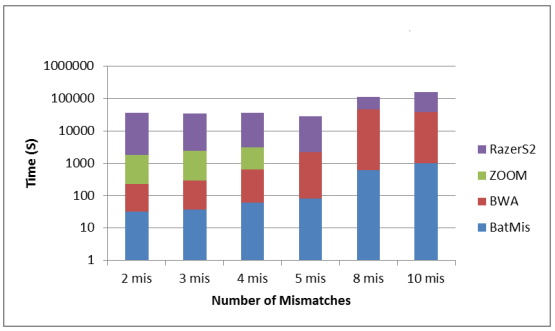
We next benchmarked the time taken by each aligner to produce multiple mappings of a given read. We define multiple mappings for a mismatch threshold k to be all the possible mappings of a read that have at most k mismatches. The 100bp real life data set used in the previous section was used to check multiple mappings. BatMis and BWA have options to scan all possible hits with less than k mismatches. For ZOOM and Razers2, such a mode is not present. Therefore, they were set to produce the first 100,000 hits for each read. ZOOM and BWA gave a small number of false hits mainly due to boundary errors. The results for the mappings are given in Table 6.3 and the time taken for the mappings are given in Figure 6.2.

	51bp			**100bp**			
	3-mis	4-mis	5-mis	4-mis	5-mis	8-mis	10-mis
BatMis	843466	856383	863707	887947	892698	903577	909360
BWA	843466	856383	863705	887947	892698	897553	889370
ZOOM	**843467**	**856384**		887947			
RazerS2	**843468**	856383	863707	**887948**	**892701**	**903579**	**909362**

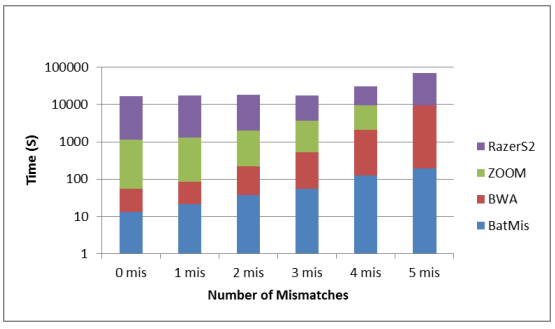
Table 6.2: *Table showing the unique mappings reported by aligners allowing different numbers of mismatches. 1 000 000 reads from the datasets ERR000577 (51bp) and ERR024201 (100bp) were mapped. All aligners report same hits upto 2 mismatches and 3 mismatches for 51bp and 100bp reads respectively. Only BatMis performs consistently across all mismatches. Other aligners report false mappings or under reports hits at high mismatches. The extra mappings in the bold entries for ZOOM and RazerS2 are due to incorrect mappings.*



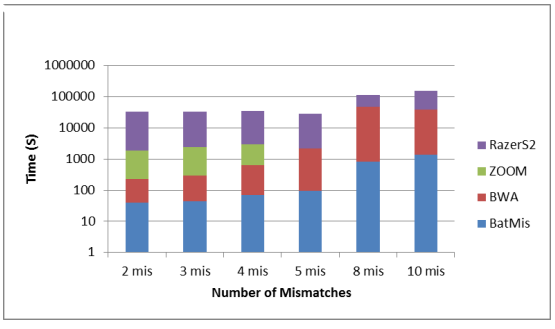
(a) Timings for 51bp least mismatch hits.



(b) Timings for 100bp least mismatch hits.



(c) Timings for 51bp unique hits.



(d) Timings for 100bp unique hits.

Figure 6.1: *Timings for searching for least mismatch hits and unique hits in the two real life datasets ERR000577 (51bp) and ERR024201 (100bp). Each library contained 1 000 000 reads. The time is shown in logarithmic scale.*

	1-mis	2mis	3-mis	4-mis	5-mis
BatMis	12709391	31001496	59121695	96502481	143315831
BWA	**12709417**	**31001544**	**59121789**	**96501738**	**143186148**
ZOOM	**12709421**	**31001568**	**59121802**	**96502628**	
RazerS2	12709391	31000921	59119338	96498204	143309572

Table 6.3: *Number of multiple mappings reported by aligners for different numbers of mismatches. 1 000 000 reads from the 100bp library ERR024201 were aligned. Bold text shows mappings that contain invalid alignments. The maximum number of invalid alignments reported is 168 by BWA at 5 mismatches. BatMis can recover all the correct mappings reported by other programs.*

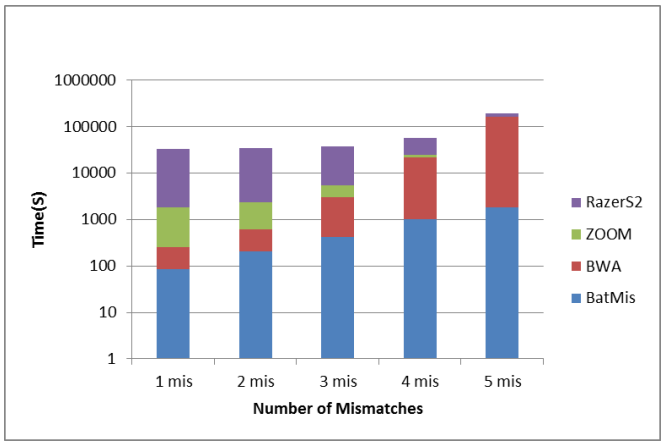


Figure 6.2: *Timings for reporting multiple mappings allowing different numbers of mismatches. 1 000 000 reads from the 100bp library ERR024201 were mapped. The time axis is logarithmically scaled. BatMis consistently reports the fastest timing.*

BatMis reports all the true hits found by other aligners and is faster than them when performing multiple mappings. All aligners report about the same number of hits, with the largest number of incorrect mappings reported being 168. The details of the incorrect mappings for each aligner is given in Table A.1 in the Appendix.

6.6.4 Comparison Against Heuristic Methods

Instead of searching for the exact solution, BWA and RazerS2 can employ heuristics to speed up mapping. BWA will first find hits in a seed region allowing at most two

mismatches, and extend the rest of the read allowing a given number of mismatches in the full read. RazerS2 has a heuristic mode where the reads can be mapped with 99% accuracy. Heuristics may miss some hits. This will result in incorrectly calling uniquely mapped reads. Table 6.4 shows the number of true unique hits recovered using heuristics modes of BWA and RazerS2 against the exact algorithm of BatMis. Table 6.5 shows the timings. The mapping procedure was similar to that in Section 6.6.2, except that BWA was run in its seeding mode and RazerS2 was run with its default sensitivity of 99%.

The results show that BatMis is much faster than RazerS2 in all cases. BWA performs very well in its seeded mode on real data for long reads. For 51bp reads, BatMis is faster than BWA and produces more mappings. For 100bp reads, BatMis is faster than BWA up to 5-mismatches. At 8 and 10 mismatches the speeds are similar, with BatMis again producing more hits. The false unique hits reported by the aligners in their heuristic modes are negligible, and a reasonable number of correct unique hits were recovered. Table A.2 in the Appendix gives details about falsely mapped reads.

Table A.4 in the Appendix shows the statistics for obtaining multiple hits with BWA in its heuristic mode and with BatMis. RazerS2 was not considered for this test as it is very slow even for finding unique hits. We can see that BatMis is still faster or has comparable speed with BWA. The heuristics of BWA can produce exact results up to two mismatches. For the 100bp reads, the heuristic mode of BWA misses between 2%-9% hits, and for 51bp reads it will miss 11%-30% hits when mapped allowing 3 to 5 mismatches.

6.7 Discussion

The solution for the k-mismatch mapping is important to sequence mapping. My algorithm BatMis can solve the k-mismatch problem exactly and efficiently. The key idea of BatMis is the recursive partitioning of the read so that a non-redundant set of

		2-mis	**3-mis**	**4-mis**	**5-mis**
51bp	BatMis	817844	843466	856383	863707
	BWA	817844	841850 **(467)**	852875 **(1400)**	858635 **(2540)**
	RazerS2	816536 **(119)**	842838 **(107)**	855989 **(185)**	863604 **(93)**
100bp	BatMis	881586	892698	903577	909 360
	BWA	881207 **(48)**	887122 **(167)**	810186 **(1592)**	904356 **(2782)**
	Razers2	881586	887947	903569 **(4)**	909347 **(22)**

Table 6.4: *Number of unique mappings reported when heuristic methods are used by BWA and Razers2. The 51bp and 100bp reads used in the previous experiments were mapped with the seeded mode of BatMis and the default alignment mode of Razers2 which can produce 99% accurate results. BatMis produces the largest number of correct hits.*

	51bp				**100bp**			
	2 mis	3 mis	4 mis	5 mis	2 mis	3 mis	4 mis	5 mis
BatMis	38	55	125	191	44	96	805	1389
BWA	191	397	438	445	257	484	764	922
RazerS2	10567	10669	11817	16049	34387	23575	26244	36688

Table 6.5: *Timings for finding unique hits when mapping reads under the heuristic modes of BWA and RazerS2. BatMis is either the fastest or has a comparable speed to the fastest aligner.*

seeds is identified for k-mismatch mapping. BatMis incrementally and systematically searches for mismatches, investigating possible k-mismatch hits found during $k-1$ mismatch scans for $k > 1$. This minimises redundant searches and provides a narrower search space for BatMis compared to the other aligners.

We checked the ability to find least mismatch hits, unique hits and multiple hits of some of the current state of the art aligners that solve the k-mismatch problem. Our results show that some aligners cannot reliably map reads with a large number of mismatches. On the other hand, BatMis was able to recover all the hits and was faster. Finally, BatMis is faster or has a similar performance when compared with the heuristic modes of other aligners. These results show that BatMis is a robust aligner that performs well at all mismatch thresholds. BatMis is a useful alternative for mapping NGS reads when we want to perform multiple mapping, unique mapping or when we want to tolerate a large number of mismatches.

Chapter 7

Alignment With Indels

7.1 Introduction

The previous chapter introduced a fast and an exact solution for mapping reads allowing mismatches. Although the BatMis algorithm can be modified to handle indels by simulating them during suffix and prefix extensions, its performance would deteriorate like the other trie based methods even when handling moderately sized indels. A long read containing structural variations will fail to align to a contiguous area in the reference genome even if mismatches and indels are allowed. In such cases, trying to map the whole read will fail and the correct approach would be a seed-based method. The aligner must map reads having a mixture of indels and mismatches and pick the best alignment out of many possible alignments. Furthermore, a modern aligner is expected to return a mapping quality reflecting the confidence of the mapping. Therefore, a good aligner must take in to consideration all these factors.

In this chapter, we introduce BatAlign, our solution for mapping long reads. The basic algorithm is designed to handle reads of around 100bp, but can be extended to handle longer reads of around 250bp. We use a seed-and-extend approach. The seeds are chosen to be 75 bp substrings of the read and mapped allowing mismatches

and indels. The indels are efficiently handled using the data structure introduced in section 3.7. In contrast to the existing aligners, a systematic scan is done to select the best seeding positions using two methods called reverse alignment and deep scan. The final extension of seeds is done using a fast Smith-Waterman method and a novel semi-global alignment algorithm. The result is a fast and accurate aligner that can produce better alignments than BWA at a speed comparable to Bowtie2.

7.2 Dynamic Programming and Sequence Alignment

Similarity between two sequences S_1 and S_2 can be determined in two ways. The first method is by globally aligning the two, i.e. determining the best alignment between the two complete sequences. The second method is by locally aligning the two sequences, where the best matching segments of the two sequences are determined. If the sequences have a high similarity, global alignment is more suitable. If the sequences are divergent, local alignment is preferred. A hybrid mode of alignment is the semi-global alignment, where global alignment is attempted, but gaps at the ends of the sequences are ignored. This method is appropriate when one sequence is made out of a portion of the other sequence.

To determine a best match objectively, an alignment score is used. Mismatches, matches and indels are given different penalties and are added together to get the alignment score. Sometimes, an affine penalty is used, where the existence of a gap is given a fixed penalty, and the extension of the gap is penalized linearly. The start of a gap is given a higher penalty and each extension of the gap is penalised less. For semi-global alignments, the start and end gaps are not penalised. The best alignment in terms of such a scoring function is deemed the best alignment.

Finding the best score for these types of alignments is generally done using dynamic programming. The standard methods for determining global and semi-global alignments is the Needleman-Wunsch algorithm [113]. The standard algorithm for determining the

local alignment is the Smith-Waterman algorithm [133]. The time complexity of these algorithms is $O(|S_1||S_2|)$. However, if the number of allowed mismatches and indels are restricted to k, "banded" versions of these algorithms can be used to bring the time complexity down to $O(k|S_1|)$, where S_1 is the query string.

7.3 The Pairing Problem

Given two patterns P_1 and P_2 of length l, we say that P_1 and P_2 have a pairing within a distance d if these patterns can be found at most a distance d away from each other, with P_1 having the least genomic co-ordinate. If x and y are respective occurrences of P_1 and P_2 within a distance d, this pairing can be represented by the ordered pair (x, y). Let $Occ_T(P)$ denote the number of times the pattern P occurs in T. We will now describe how the data structure $L_{T,l,\delta,D,\kappa}$ described in Section 3.7 allows us to efficiently find pairings of P_1 and P_2 that are within a distance d, and when $Occ_T(P_1) < D$ and $Occ_T(P_2) < D$. Precisely, we have the following cases;

1. If $Occ_T(P_1) < \delta$ and $Occ_T(P_2) < \delta$, we find all locations of P_1 and P_2 using Algorithm 3.5. Then these locations can be cross checked to obtain all pairings of P_1 and P_2.

2. If $Occ_T(P_1) < \delta$ and $Occ_T(P_2) \geq \delta$, we obtain all the locations of P_2 from $L_{T,l,\delta,D,\kappa}$. Since they are sorted by their position, we can check to see if any occurrence of P_1 (found using Algorithm 3.5) can be paired with an occurrence of P_2 by performing a binary search.

3. If $Occ_T(P_1) \geq \delta$ and $Occ_T(P_2) < \delta$, this case can be solved similar to the previous case, by looking up all the occurrences of P_1 from $L_{T,l,\delta,D,\kappa}$ and pairing them with all the occurrences of P_2 (found using Algorithm 3.5).

4. If $Occ_T(P_1) \geq \delta$ and $Occ_T(P_2) \geq \delta$, all the occurrences of P_1 and P_2 are obtained

from $L_{T,l,\delta,D,\kappa}$. These two lists are merge sorted. If an occurrence of P_1 is followed by an occurrence of P_2, and they satisfy the distance constraint, they are reported.

Note that if either $Occ_T(P_1) \geq D$ or $Occ_T(P_2) \geq D$, then all of their occurrences can be found using Algorithm 3.5 and sorted by genomic locations, and one of the above methods can be applied to find the pairings. This process is slow. However, these cases would be rare if D is chosen sufficiently large, and can be ignored in practice.

7.4 Mapping Reads With Indels

We will now discuss a method of mapping a read having a single indel. If R is the read, the indel might be completely contained in its left half R_l or the right half R_r. The other possibility is that the indel might be present in both the left and the right halves of R. In the former cases, the indel can be found by using R_l and R_r as seeds, and extending the seed locations allowing an indel. This method can identify an indel having a maximum size of $\lceil |R|/2 \rceil$. For the latter case, we will make use of the pairing algorithm described in section 7.3.

We choose $P_1 = R[1, l]$ and $P_2 = R[|R| - l + 1, |R|]$, i.e., P_1 and P_2 are the prefix and suffix l-mers of R respectively. If there is an indel of size less than or equal to d between P_1 and P_2, we should be able to find a pairing of P_1 and P_2 within a distance $d+l$. Once all such locations are identified, they can be verified. This method is capable of identifying an indel having a maximum size of $|R| - 2l$. Overall, we can detect indels of size $\min(\lfloor |R|/2 \rfloor, |R| - 2l)$.

Algorithm 7.1 gives the full details of mapping a read allowing k mismatches and one indel. The only difference from the previous description is that R_l, R_r, P_1 and P_2 are mapped allowing upto k mismatches. Furthermore, care is taken in Line 10 so that the total number of mismatches in P_1 and P_2 does not exceed k.

Algorithm 7.1: Find_Indel($R, T, k, d, L_{T,l,\delta,D,\kappa}$)

Data: A read R, a reference T, a mismatch threshold k, indel size d and the data structure $L_{T,l,\delta,D,\kappa}$

Result: Alignments of R having at most k mismatches and an indel of size less than or equal to d

`/* Case 1: Indel is in one half of the read */`

X_1= The set of k mismatch hits of $R[1, R|/2]$ in T;

H_1= The set of hits having k mismatches and one indel obtained by extending hits in X_1;

X_2= The set of k mismatch hits of $R[|R|/2 + 1, R|]$ in T;

H_2= The set of hits having k mismatches and one indel obtained by extending hits in X_2;

`/* Case 2: Indel is in both halves of the read */`

for $j = 0$ *to* k **do**

    X_j= The set of hits of $R[1, l]$ in T having j mismatches;

    Y_j= The set of hits of $R[|R| - l + 1, R|]$ in T having j mismatches;

end

for $j = 0$ *to* k **do**

    $H = H \cup \{$ Pairings of X_j and Y_{k-j} that are within a distance $l + d.\}$;

end

$Hits$ = The set of hits in H that can be aligned to have k mismatches and one indel;

return $H_1 \cup H_2 \cup Hits$.

7.5 Reverse Alignment

The basic approach of seed based aligners is to first map the seeds allowing a low number of mismatches, and then after a possible elimination of some of these seeds (e.g. based on repetitiveness, after too many extensions etc.) the rest of the seeds are extended. This approach could possibly filter out the correct alignment, resulting in an incorrect mapping. To minimize this problem we use a strategy called reverse-alignment. Reverse-alignment searches for the best possible alignment first. This is in a sense the reverse of what seed based aligners do, as they first map the seeds and then search for the best alignment.

Given a read R, reverse-alignment incrementally finds the hits of R with the most likely combination of mismatches and indels. We define a function F as $F(i) = (p_i, q_i)$, where i, p_i and q_i are non-negative integers. p_i and q_i represent the number of mismatches and indels in an alignment respectively. This function determines the likelihood of a given mismatch/indel combination. If $a < b$, then the probability of the correct alignment of a read having a (p_a, q_a) mismatch/indel combination is higher than that of having a (p_b, q_b) mismatch/indel combination. Reverse-alignment incrementally tries to map R allowing $F(i)$ mismatch/indel combinations for $i = 1, 2, \ldots$. We will describe how to define F in the following section.

7.5.1 Determining F

In real-life, the likelihood of an indel in a genome is an order of magnitude less than that of a SNV (Single Nucleotide Variation). The likelihood of finding multiple indels within a read becomes small as the length of R becomes small. If the sequencing is error-free, we can expect mismatches in a read R to appear at a rate equal to the expected number of SNVs in a segment of length $|R|$ in the reference genome. However, empirical studies show that, for Illumina and SOLiD reads, the majority of mismatch errors are due to sequencing errors. A general heuristic for such platforms is to set the

mismatches in a read due to sequencing errors and/or SNVs to be about $\sim 5\%$ of the read length [74]. Furthermore, indels occur at a rate of $\sim 0.02\%$ [157] in the human genome . Based on these statistics, for a read of length around 75 bp, we can set one indel and four mismatches as a reasonable upper bound for the number of indels and mismatches to be allowed in a mapping. The definition of F used by default is to set $F(i)$ such that $F^{-1}(n_1, n_2) < F^{-1}(n_1 + 1, n_2)$ for $n_1 \leq 4$ and $F^{-1}(5, 0) < F^{-1}(0, 1)$ for $i = 1, 2 \ldots 9$. That is, we set $F(1) = (0, 0), F(2) = (1, 0), F(3) = (2, 0), F(4) = (3, 0), F(5) = (4, 0), F(6) = (5, 0), F(7) = (0, 1), F(8) = (1, 1)$ and $F(9) = (2, 1)$.

7.6 Deep-Scan

During reverse-scanning, in terms of mismatches and indels, the combination $F(i)$ is better than that of $F(i + 1)$. However, it may be possible that, when the quality information is taken into account, the hits having $F(i + 1)$ mismatches and indels are the better ones. To give an example, suppose there is a two mismatch hit with both the mismatches appearing at very high quality base, and a three mismatch hits whose mismatches are at very low quality bases. We would want to prefer the latter over the former as the best hit if the likelihood of having two SNV's in the read is small for the read length and the reference genome under consideration. Because of this reason, when we want to determine the best hits, it is always better to consider hits that are similar to the best hits as determined by the function F. We can do this by reporting all the hits $F(j)$, where $F(i)$ is the best mismatch/indel combination for R and $i < j \leq 9$. However, this scheme would be too time consuming to implement in practice, and we use the following strategy. If there are multiple hits having $F(i)$ mismatch/indel combination, we stop the reverse-alignment scanning. Otherwise, if $i < 9$, we scan one level deeper and report all the hits having $F(i + 1)$ mismatch/indel combination.

7.7 Quality-Aware Alignment Score

Sequencing data usually produce a quality score that indicates the reliability of bases called at each position of a read. If the probability of the i^{th} base of a read being correct is $P(i)$, the quality score $Q(i)$ assigned to that base is given by $P(i) = 1 - 10^{-Q(i)/10}$. Assuming that there is no bias to a particular set of nucleotides when a base gets miscalled with another nucleotide, the probability of the i^{th} base of a read being miscalled is $1 - P(i)/3$.

For a given alignment, we can compute a quality-aware alignment score based on an affine scoring scheme as follows. The score for a match or a mismatch at $R[i]$ is the phred scaled value of $P(i)$ (i.e. $-10 \log P(i)$). When indels are present, positive gap open and extend values are chosen. These values are the Phred scaled probabilities calculated for gap openings and extensions using data from human resequencing projects by Novoalign (www.novocraft.com). Based on this scoring scheme, lowest scoring alignments are the most likely alignments.

7.8 The BatAlign Algorithm

In the previous chapter we have described how to find mismatch hits incrementally, and the Algorithm 7.1 shows how to find indels allowing mismatches. In fact, $Find\_Indel$ can be unrolled and re-ordered so that indel hits having K mismatches are reported before those having $K + 1$ mismatches. Therefore, reverse-scan can be easily implemented.

The BatAlign algorithm for mapping reads having around 75 bases is described in Algorithm 7.2. It has two basic steps. In the first step it gathers a likely set of candidate hits. In the second stage, if there is a unique alignment it is reported. Otherwise, for each of the candidate hits its quality-aware alignment score is found. If the difference between the two lowest quality-aware alignment scores is greater than some cutoff (set at 10 by default) the lowest scoring alignment is reported. If this cutoff is set high, we

can be more confident that the hit having the lowest quality-aware alignment score is more likely to be the correct hit than the next best alignment.

Algorithm 7.2: BatAlignShort(R, T, F, M, G)

Data: A read R, a reference T, a reverse-alignment function F where $F(i)$ is defined for $i = 1, \ldots, M$, the threshold for reporting multiple hits G
Result: The best alignment for R

```
   /* Step 1:  Gather candidate hits                                    */
 1 C = ∅;
 2 for i = 0 to M do
 3     H1=Hits of R in T having F(i) mismatch/indel combinations;
 4     if H1 is not empty then
 5         if H1 contains unique hit and i < M then
 6             H2=Hits of R having F(i + 1) mismatch/indel combinations;
 7             C = H1 ∪ H2;
 8         else
 9             C = H1;
10         end
11     end
12 end
   /* Step 2:  Report hit                                              */
13 if C contains a unique hit then
14     Report unique hit in C;
15 else
16     Find alignment scores of hits in C;
17     if alignment score difference of the two lowest scoring hits < G then
18         Report the lowest scoring hit;
19     end
20     Report no alignment;
21 end
```

When aligning reads longer than 75 bases, the read will be divided into as many non-overlapping 75bp seeds as possible. Precisely, when $|R| > 75$, the seeds for the read R are given by $R[75i + 1, 75(i + 1)]$, where $i = 0, \ldots, \lfloor |R|/75 \rfloor$. Each of these are aligned to the genome with the *BatAlignShort* algorithm. If the top hit of two of these seeds fall into the neighbourhood of each other, the whole read is aligned to the neighbourhood of the leftmost seed. Otherwise all the candidate hits found by the seeds

are extended, and the lowest scoring alignment is reported if the difference of the two lowest scoring alignments is larger than 10.

7.9 Extension of Seeds

Seed verification needs to be done in the $Find\_Indel$ algorithm. For small read lengths, we can expect the neighbourhood where seeds map to be similar to the read. Therefore, semi-global alignment is a good choice for seed extension. We have developed a fast semi-global alignment algorithm that can be used to extend seeds allowing at most one indel and a small number of mismatches. When the alignment score of the semi-global alignment drops below 90% of the maximum alignment score (i.e. the score for an exact match), a Smith-Waterman alignment is done, as it may be able to find a better local alignment in these cases. We use a fast implementation of the Smith-Waterman method that uses SIMD and SSE2 instructions [158] based on the Farrar's method [34]. This algorithm determines the best alignment in two steps: First it will calculate the best Smith-Waterman score and then it will perform a banded alignment to get the trace-back of an optimal alignment from the DP-table.

When compared with a version that completely relies on the Smith-Waterman method for seed extension, we have noticed a 30% speed up of BatAlign's running time when the fast semi-global alignment is used in conjunction with the Smith-Waterman method. Next, we will describe this algorithm.

7.10 Fast Method for Seed Extension

In this section, we will describe a method faster than the Smith-Waterman method that can be used to extend seeds, assuming only one indel exists in the best alignment. We will use the following terminology. Define μ_1 and μ_2 to be the mismatch and match penalties respectively, and let $GP(d)$ be the gap penalty for an indel of size d. Let

$H(X_1, X_2)$ be the Hamming distance and $AS(X_1, X_2)$ be the semi-global alignment score between the two strings X_1 and X_2 calculated using penalties μ_1, μ_2 and GP. Finally, when $|X_1| = |X_2|$ and the alignment between them does not contain gaps, define $\mathrm{AS}_{\mathrm{Ham}}(X_1, X_2) = \mu_1 H(X_1, X_2) + \mu_2(|X_1| - H(X_1, X_2))$. Intuitively, $\mathrm{AS}_{\mathrm{Ham}}(X_1, X_2)$ is the global alignment score of an alignment that does not have indels in it. We will assume that the penalties are chosen so that the best alignment will have the best alignment score.

7.10.1 Special Case of Alignment

We will first introduce an algorithm to efficiently compute the alignment of two strings R and G assuming that in their best alignment (1) there is a maximum of one indel whose size does not exceed D bases, (2) the number of mismatches do not exceed M and (3) there are no gaps at the start and the end of the alignment. If the indel appears after the k^{th} base in G and R, the alignment score of such an alignment is given by $\mathrm{AS}_{\mathrm{Ham}}(G[1,k], R[1,k]) + GP(d) + \mathrm{AS}_{\mathrm{Ham}}(G[k+1+d_1, |R|+d_1-d_2], R[k+1+d_2, |R|])$. In this formula, if there is an insertion of size d_1 in R, we set $d = d_1$ and $d_2 = 0$. If there is a deletion of size d_2 in R, we set $d = d_2$ and $d_1 = 0$. We will describe our algorithm for the case where an insertion is present in R.

Algorithm 7.3: Scan_Insert_Sizes(R, G, D, M)

Input: two strings R and G, a maximum insert size D, a maximum mismatch number M.
Output: *Insert_Sizes* returns possible sizes of inserts that can occur in $R[1, |R|/2]$.

```
1 Insert_Sizes = ∅;
2 for i = 1 to D do
3     if Hamming distance between G[|R|/2 + 1 + i, |R| + i] and R[||R|/2 + 1, |R|]
      does not exceed M then
4         Add i to Insert_Sizes;
5     end
6 end
7 return Insert_Sizes
```

Algorithm 7.4: Get_Best_Hit($R, G, Insert\_Sizes$)

Input: two strings R and G, list of possible sizes of inserts in $R[1, |R|/2$
$Insert\_Sizes$
Output: Insert having the best score, along with its size and location in R

```
1  Max_Score = −∞;
2  foreach i in Insert_Sizes do
3      for x = 1 to |R|/2 do
4          Score = alignment score of R and G[1, |R| + i] with an insertion of size i
           at position x;
5          if Score > Max_Score then
6              Best_Hit = (i, x, Score);
7              Score = Max_Score;
8          end
9      end
10 end
11 return Best_Hit
```

We will start with a solution to find inserts in the left half of R. The key idea is simple. If an insertion of size i exists in the left half of R, then the right half $R[|R|/2 + 1, |R|]$ must align to $G[|R|/2 + 1 + i, |R| + i]$ with a Hamming distance less than M (See Figure 7.1(B)). Such an alignment narrows down the region an insertion can occur, and the exact position of insertions can be then verified by examining this region. Procedure $Scan\_Insert\_Sizes$ described in Algorithm 7.3 returns the possible sizes of inserts that satisfy this condition. It does so by trying all the possible values i for the size of an insert (Line 2) to see if the right half of R can be mapped to $G[|R|/2 + 1 + i, |R| + i]$ allowing at most M mismatches (Line 3).

Next, we will show how to find the most likely location for an insertion if its size is known. Note that if the insertion is the region $G[x + 1, x + i]$, then $G[1, x]$ and $G[x + i + 1, |R| + i]$ must align to $R[1, x]$ and $R[x + 1, |R|]$ respectively (see Figure 7.1(B)). The corresponding alignment score is given by $\mathrm{AS}_{\mathrm{Ham}}(R[1, x], G[1, x]) + GP(i) + \mathrm{AS}_{\mathrm{Ham}}(R[x + 1, |R|], G[x + i + 1, |R| + i])$. We can find the most likely location for an insertion by finding the location that maximizes this alignment score.

Based on this idea, the procedure Get_Best_Hit given in Algorithm 7.4 takes a list of possible insert sizes, and reports an optimum alignment having an insert in the left half of R. Taking a possible size of an insertion i from the list $Insert\_Sizes$, and assuming that the insert is located after the x^{th} base, Get_Best_Hit will calculate its alignment score (Lines 2-4). Finally, it will find a best alignment along with the score (Lines 5-8) and return it. Note that the location of the insertion x and the size of the insertion i are sufficient to specify the alignment.

We now have a method to detect a best alignment containing an insertion in the left half of a string. We can use this method recursively to find all the insertions in R as follows. The insertions can be in either the left or the right halves of a string R. First, we look for insertions in the left half of R. Next, if the left half of R can be aligned with $G[1, |R|/2]$ allowing $M'(< M)$ mismatches, we recursively look for insertions using the same method by shifting R and G by $|R|/2 + 1$ bases to the right, and allowing $M - M'$ mismatches.

Using this technique, the function $Fast\_Insert$ described in Algorithm 7.5 returns an optimum alignment containing an insertion for a read R under the given criteria.We will define $S_j = |R| + 1 - \lceil |R|/2^j \rceil$, and $S_0 = 1$ for a non-negative integer j. In the j^{th} iteration, $Fast\_Insert$ will be processing the substring $R[S_j, |R|]$, which is the right half of the string $R[S_{j-1}, |R|]$ for $j > 1$. The function $Fast\_Insert$ starts its j^{th} iteration by selecting the substring $R' = R[S_j, |R|]$ and $G' = G[S_j, |G|]$ from R and G respectively (Line 2). Then it finds an optimum alignment of R having an insert in R' (Lines 6-7), and calculates its alignment score (Line 8). Next, to search for an insertion in the right half of the string, we first check if $H(R[1, R|/2], G[1, R|/2])$ does not exceed M. If this condition is satisfied, we recursively search for an insertion in the right half R' (Line 11). The best scoring hit is then returned (Lines 12-13,16). The recursion stops when the string consists of a single base (Line 3). We can similarly write a procedure $Fast\_Delete$ to return an optimum alignment having a deletion.

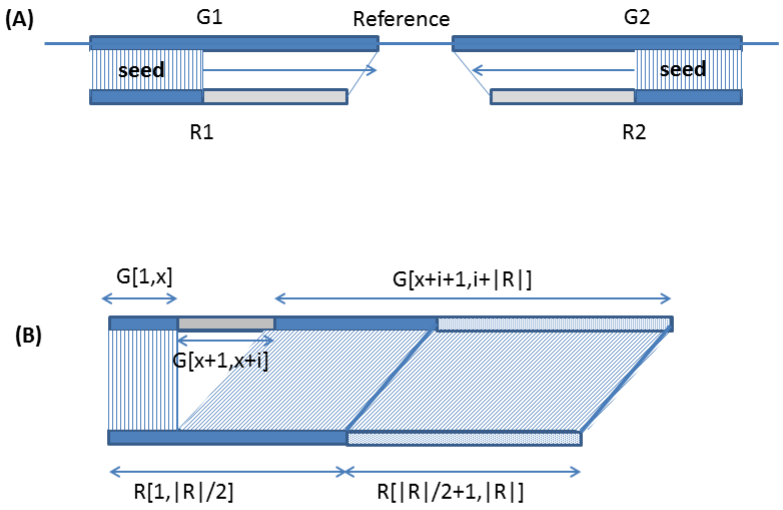


Figure 7.1: *(A) When performing seed extension, the seed portion of the reads $R1$ and $R2$ will be aligned to the genome. The seed will be on the left half of $R1$ and the right half of $R2$. Neighbourhood of the area seeds mapped to will then be taken; $G1$ to the right of $R1$, $G2$ to the left of $R2$. Semi-global alignment can then be done between $R1 - G1$ and $R2 - G2$, which will extend the seeds to right and left respectively. This semi-global alignments can have gaps at only the right and left ends respectively. (B) When the read R contains an insert in the left half of R after the x^{th} base, the right half of the read can be mapped completely to the reference as shown.*

Algorithm 7.5: Fast_Insert(R, G, D, M, j)

```
1  if j = 0 then S_j = 1 else S_j = |R| + 1 − ⌈|R|/2^j⌉;
2  R' = R[S_j, |R|]; G' = G[S_j, |G|];
3  if length of R' ≤ 1 then
4      return (0,0,−∞)
5  end
   /* Find inserts in left half of R                                  */
6  Insert_Sizes = Scan_Insert_Sizes(R', G', D, M);
7  (i_l, x_l, Score_l) = Get_Best_Hit(R', G', Insert_Sizes);
8  if j ≠ 0 then Score_l = Score_l + AS_Ham(R[1, S_j − 1], G[1, S_j − 1]);
   /* Find inserts in right half of R                                 */
9  M' = H(R[1, S_{j+1}], G[1, S_{j+1}]);
10 if M' < M then
11     (i_r, x_r, Score_r) = Fast_Insert(R, G, D, M − M', j + 1);
12     if Score_r > Score_l then
13         return (i_r, x_r, Score_r)
14     end
15 end
16 return (i_l, x_l, Score_l)
```

7.10.2 Semi-Global Alignment for Seed Extension

When a seed extension is performed in BatAlign, a part of the read will be anchored and the rest of the read will be extended (See Figure 7.1 (A)). Then the read alignment will be a variant of the semi-global alignment with gaps being permitted in only one side of the read. We will now discuss how to solve this problem. We will assume the case where the left part of the read is anchored and is extended from left to right. In this case, a gap is permitted only in the rightmost end of the read.

To solve this problem, we need to find a best alignment among (1) alignments containing an indel at the rightmost end (2) alignments containing an indel in the middle of R. The second case can be solved with the routines $Fast\_Insert$ and $Fast\_Delete$. For the first case, we can find all the possible cases of alignments having an indel at the end and pick the best one. However, any indel at the end of R will not be penalized in a semi-global alignment, and therefore any indel appearing after R will not contribute

to the alignment score. In fact, we need to consider only the cases of insertions where an insert of length i ($i = 0, 1, \ldots, D$) appears in the region of $R[|R| - i + 1, |R|]$. Their alignment score will be simply $\mathrm{AS}_{\mathrm{Ham}}(R[1, |R| - i], G[1, |R| - i)$.

Procedure *Find_Opt_Right* described in Algorithm 7.10.2 returns a best alignment based on the above discussion. First it will find the best alignment with an indel at the rightmost end of R by calculating all possible distinct alignment scores (Lines 2-7). Next, it will find optimal scores allowing an indel in the middle of the read using $Fast\_Insert$ and $Fast\_Delete$ routines(lines 8-9), and finally returns the alignment having the best score (Line 10).

Algorithm 7.6: Find_Opt_Right(R, G, D, M)

```
1  Mis_Score = −∞;
2  for j = 0, 1, ..., D do
3      M = AS_Ham(R[1, |R| − j], G[1, |R| − j]);
4      if M > Mis_Score then
5          (i_m, x_m, Score_m) = (|R|, j, Mis_Score);
6      end
7  end
8  (i_i, x_i, Score_i) = Fast_Insert(R, G, D, M, 1);
9  (i_d, x_d, Score_d) = Fast_Delete(R, G, D, M, 1);
10 Report the alignment with largest score among Score_m, Score_d and Score_i;
```

A similar algorithm can be constructed for the case where the seed is taken from the right side of the read, and the extension is performed from right to left.

7.11 Proof of Correctness

We will first prove that $Fast\_Delete$ and $Fast\_Insert$ will return an optimum alignment when the indel is in the middle of the read.

Lemma 7.11.1. *Algorithm $Fast\_Insert$ returns an optimum alignment having M mismatches and one insertion whose length does not exceed D in the middle of R.*

Proof. The recursion will stop when $j = \lg |R|$, so the algorithm will terminate. The j^{th} Iteration of $Scan\_Insert\_Sizes$ will return all possible sizes of insertions that are allowed within $R[S_j, S_{j+1} - 1]$. Get_Best_Size will then find all possible alignments having an insertion of these sizes within $R[S_j, S_{j+1} - 1]$, and return an optimal one. Since $\cup_{j=0}^{\lg |R|} R[S_j, S_{j+1} - 1] = R[1, |R| - 1]$, all alignments having an insertion in the middle of R will be considered, and an optimal alignment will be returned. □

Lemma 7.11.2. *Algorithm $Fast\_Delete$ returns an optimum alignment having M mismatches and one deletion whose length does not exceed D in the middle of R.*

Proof. Proof is similar to that of Lemma 7.11.1. □

Lemma 7.11.3. *Algorithm $Find\_Opt\_Right$ returns an optimum semi-global alignment of R allowing M mismatches, one indel whose length does not exceed D and no gaps in the leftmost end.*

Proof. According to the conditions there are three possibilities; (1) There is no indel, (2) the indel appears in the middle of the read or (3) the indel is at the rightmost end of the read. Since an indel at the end of a semi-global alignment will not result in any penalty, Lines 2-7 will find the first and the last cases. The other case is handled by $Fast\_Delete$ and $Fast\_Insert$ as proven by Lemmas 7.11.1 and 7.11.2. □

7.12 Complexity of the Algorithm

Before discussing the complexity of the algorithm, we will show a faster way of calculating the alignment score given in Line 4 of Get_Best_Hit. As it is, the time complexity of the calculation is $O(|R|)$. However, with a pre-calculation for each i, it can be done in constant time. Let $H_l(x) = \text{AS}_{\text{Ham}}(R[1, x], G[1, x])$ and let $H_r^i(x) = \text{AS}_{\text{Ham}}(R[x + 1, |R|], G[x + i + 1, |R| + i])$. If we pre-calculate $H_l(x)$ for $x = 0, \ldots, |R|/2$ and pre-calculate $H_r^i(x)$ for $i = 1, \ldots, |R|/2$ and $x = 1, \ldots, D$, then we can see that the alignment

score in line 6 can be calculated as $H_l(x) + H_r^i(x) + GP(i)$ in constant time. The time complexity of the pre-calculation step is $O(|R|)$.

Lemma 7.12.1. *The worst case running time of Get_Best_Hit is* $O(D|R|)$.

Proof. we pre-calculate $H_l(x)$ for $x = 0, \ldots, |R|/2$ at the start of the routine. Before entering the loop in Line 3, we pre-calculate $H_r^i(x)$ for $i = 1, \ldots, |R|/2$ and $x = 1, \ldots, D$. Then the calculation in Line 4 can be done in $O(1)$ time. So the worst case time complexity of the whole function is $O(D|R|)$. □

Lemma 7.12.2. *The worst case running time of Fast_Delete and Fast_Insert is* $O(D|R|)$

Proof. The time taken for $Scan\_Insert\_Size$ is $\alpha_1 D|R'|$ for some $\alpha_1 > 0$. The worst case time taken for Get_Best_Hit is $\alpha_2 D|R'|$ for some $\alpha_2 > 0$. Therefore, the worst case time taken for $Fast\_Insert$ can be written as

$$T(|R'|) = \alpha D|R'| + T(|R'|/2)$$

for some $\alpha > 0$ when constant time factors are excluded. The solution for this recursion is $T(|R'|) = 2\alpha D|R'|$; i.e., the time complexity is $O(D|R|)$. Similar arguments can be used to show that the worst case running time of $Fast\_Delete$ is $O(D|R|)$. □

Lemma 7.12.3. *The worst case running time of Find_Opt_Right is* $O(D|R|)$

Proof. The running time of Lines 2-7 is $O(|D|)$. Similar to $Fast\_Insert$, $Fast\_Delete$ can be shown to have a worst case running time of $O(D|R|)$. Therefore the running time of $Find\_Opt\_Right$ is $O(D|R|)$. □

The complexity of the worst case running time of the fast semi-global alignment is similar to that of a banded semi-global alignment. However, in most cases the lists $Delete\_Sizes$ and $Insert\_Sizes$ will return much less that D elements as assumed in

the worst case analysis. If the best alignment is a mismatch alignment the chance of these lists being empty is high. Furthermore, there is a high chance that if the indel is actually in the left half of the read, the tests at Line 11 of $Fast\_Delete$ and $Fast\_Insert$ will fail. Therefore, in practice, we can expect a much faster execution time than a banded approach.

7.13 Increasing Sensitivity, Accuracy and Speed

In Section 7.5.1 we described our basic criteria for finding a suitable set of hits. This criteria will miss enumerating the correct hit under three cases. (1) The read contains mismatches and indels outside the scope the algorithm is set to handle. (2) The read contains only m mismatches, but has a multiple mapping with less than m mismatches or a unique mapping with less than $m-1$ mismatches. (3) A read containing indels with m mismatches might get mapped as a pure mismatch hit or as an indel hit having less than m mismatches. We use several methods to recover these three kinds of hits.

During the seed extension stage, we might recover possible hits having more than the number of mismatches and indels specified by the basic algorithm. Instead of discarding them we will add them to the pool of candidate hits. If a hit found in the mismatch scanning stage turns out to have an indel when the full-read extension is done, an indel scan is forced. If the best alignment score happens to be larger than that of a single indel hit, again an indel scan is forced. Finally, for reads having more than one mismatch, the indel scanning is performed if the average quality of bases having a mismatch is higher than that of the average base quality of the read. We allow one high quality mismatch to consider the possibility of a SNP.

7.13.1 Making the Algorithms Faster.

We can use some strategies to speed-up the algorithms. The general idea is to limit the number of hits enumerated in the deep-scan step and to avoid time consuming extension

by Smith-Waterman method. These methods will result in a loss of accuracy and sensitivity (For definition see Section 7.17.2). When extending the candidate hits we can terminate the extension process if the extensions have not resulted in an improvement in the alignment score for K successive steps. Furthermore, when an indel scan is forced, the extension can be limited to those hits already having an indel. The most time consuming part of our Smith-Waterman library is the trace-back step to get the exact alignment. We can perform this step only on most promising extensions based on the alignment score.

The methods in this section have been used in various combinations to produce several modes of scanning that trade speed with accuracy and sensitivity.

7.14 Calculating the Mapping Quality

According to the SAM format definition, an aligner should output a mapping quality (MapQ) calculated by the formula $-10\log$(probabilty of mapping position being wrong), rounded to the nearest integer. In order to calculate this probability, we need to know the probability that the read did not originate from the reference, the probability of the read being incorrectly mapped and the probability that the aligner failed to enumerate the hit etc [77, 89]. However, these probability cannot be calculated in practice and various approximations are used.

The seminal paper describing mapping qualities [77] provided a formula for practical calculation of mapping qualities, given by

$$\min\{q_2 - q_1 - 4.343\log n_2, 4 + (3-k)(\bar{q} - 14) - 4.343\log p_1\},$$

where q_1 and q_2 are the sum of the base qualities at the mismatching bases of the top and the second best hits, n_2 is the number of hits having the same number of mismatches as the second best hit, k' is the number of mismatches in the first 28bp read,

$\bar{q}$ the average base quality of the same region and P_1 is a certain probability calculated during alignment. A popular approximation is the one used in BWA-SW, which uses the empirical formula MapQ=$250 \cdot c_1 c_2 (S_1 - S_2)/S_1$, where S_1 and S_2 are the alignment scores of the top and second best alignments respectively, and c_1 and c_2 are values that depend on the nature of the alignments. Bowtie returns a MapQ deduced from simulated datasets. This value is based on the best and the second best alignment scores and the nature of the alignment. However Rufallo et. al. [121] showed that these approximations are far away from the theoretical mapping qualities. Therefore, in practice, we can think of the mapping quality as a number that measures the accuracy of the alignment, with 0 corresponding to an alignment on which we have no confidence in.

We base our mapping quality along this line of thinking and take three factors in to account. The first factor is the alignment score of the top hit and the second best hit (if it exists). If the difference of the alignment scores between the top hit and the second best hit (when it exists) is more than 10, we assign the hit a mapping quality of 0. The reasoning behind this is that the best and the second best alignment are too similar to designate one hit as the best hit.

The second factor is the Smith-Waterman alignment score of the alignment. We give an alignment score S_{ref} to the final alignment to measure the likelihood of the sequence appearing in the reference. In this alignment score, gaps are given an affine penalty. Each mismatch is given a penalty between a minimum of M and a maximum of N scaled on the quality of the mismatch base [69]. This scaling is done according to the formula

$$M + (N - M)(\min(Q, 40)/40),$$

where Q is the mapQ of the mismatching base. M and N are set at 6 and 2 [69]. A mismatch on a higher quality base will incur a higher penalty. If this value is high, we can be more confident about the alignment. The final factor is the repetitive nature

of the read. During the alignment stage, we count how many alignments S_{multi} the first half of the read will have when 0 mismatches are allowed. A larger value of S_{multi} indicates a higher probability that the read will align merely by chance. However, to avoid over penalizing highly repetitive alignments we limit S_{multi} to a maximum of 10.

The final mapping score is calculated by the formula $60 + S_{ref} - S_{multi}$. The mapping quality is capped at 60, we set it to be 60. If the mapping quality is negative we set it to zero.

7.15 Results

We will first compare the results of our method BatAlign against a set of state of the art aligners that have been published. The aligners compared were BatAlign (r169), Bowtie2 (2.0.6), BWAShort, BWA-SW (0.6.1-r104), Stampy (v1.0.16), GEM (3rd release) and SeqAlto (0.5-r123). Bowtie2 is the de facto aligner to benchmark fast aligners. BWAShort and BWA-SW are the de facto aligners to benchmark for accurate short read and long read mappings respectively. Stampy is very sensitive in the presence of variations, and GEM is the fastest aligner available. Seqalto is another fast aligner that is hash-based. These aligners feature a wide range of mapping techniques. We ran Stampy standalone without using BWA as this mode is reported to be more sensitive. We also ran Bowtie2 in the `--local` mode to increase sensitivity. All the other settings were the default settings for all the programs. All the benchmarked aligners except GEM produce a quality score. For each tool, the reference genome was indexed with default indexing parameters. hg19 was used for all the experiments in this chapter. All experiments were run on a Linux workstation equipped with Intel X5680 (3.33 GHz) processor and 16GB RAM.

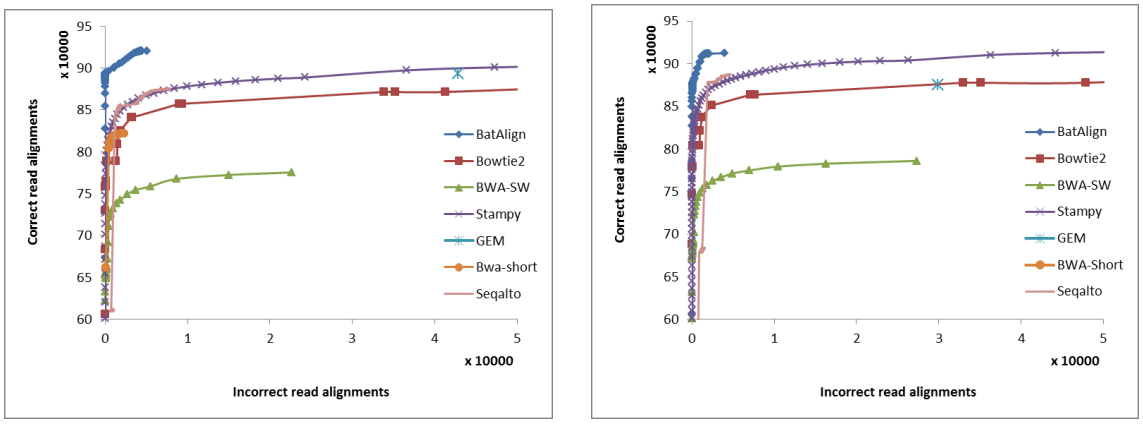
7.16 Simulated Data

We will first present our comparison on a simulated datasets whose point of origin in the genome are known. We generated two classes of simulated data. The first class produced Illumina-like reads while the second class had one indel in each of its reads. The first class of reads were generated using the ART simulator [51] excluding the non-chromosomal sequences of hg19. We have chosen ART since the substitution errors are simulated according to empirical, position-dependent distribution of base quality scores; it also simulates insertion and deletion errors directly from empirical distributions obtained from the training data derived from the 1000 genome project [25]. Empirical read quality score distributions are provided by ART for read lengths of 75 bp, 100 bp and 250bp. The number of mismatches and indels (SNVs, base-call errors or gaps) in a simulated read was capped at 7% (an indel was counted as one error).

The second class of pure-indel reads was used to demonstrate the performance of BatAlign on the recovery of indels. 16 sets of 75 bp reads were created. Each pure-indel data set contained 1 million reads having one indel of a fixed length (since the average density of an indel is one in 7.2kb of DNA [106]). Indel lengths ranged from 1 bp to 8 bp and each data set contained either insertions or deletions only.

7.17 Evaluation on ART-Simulated Reads

For this evaluation, we discarded mappings with mapQ = 0 for all methods as they are considered ambiguous by the aligner. We then calculated the cumulative number of correct and wrong alignments and drew the ROC curve thresholded by the mapQ. Due to the inability to align some indels to their exact locations and the presence of soft-clippings (i.e. segments at the ends of the reads that have been ignored in the alignment), an alignment will be considered as a correct mapping if the leftmost position was within 50bp of the position simulated by the simulator on the same strand. If the



(a) ROC for 75 bp **(b)** ROC for 100bp

Figure 7.2: *Mapping of simulated datasets containing reads of length 75bp and 100bp. The reads were generated allowing 7% errors in a read, and a deviation of 50bp from the exact origin of the read was allowed to account for alignment errors and clippings.*

margin of error is decreased, it will improve the ROC of BatAlign even more compared to the others. However it will misrepresent aligners that can recover short fragments of a read. To be unbiased, we set the margin of error at 50 bp as chosen in the benchmarks for Stampy and Seqalto. For a further discussion on this setting see [110]. Figure 7.2 shows the ROC curves for each method and dataset. BatAlign was more sensitive and reported less wrong alignments than the other methods over a large range of mapping quality cutoffs.

7.17.1 Multiple Mappings

For structural variation and copy number variation studies, multiple hits are of interest. In this section we compare the multiple hits output from each aligner. We had all the methods report the top 10 hits for each read (Stampy can only report at most 1 hit). Since all the compared methods report hits of a read ranked by their likelihood of being the correct hit, we report the correct hits broken down by their rank. The results are shown in Table 7.1.

We can see that BatAlign recovered the most number of correct hits within its

		Rank of hits										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	Total
75bp	BatAlign	933560	11604	732	622	745	335	146	116	90	68	948018
	Bowtie2	866410	10873	4095	1541	551	314	192	142	112	77	884307
	BWA-SW	786309	2	0	0	0	0	0	0	0	0	786311
	Stampy	902757	-	-	-	-	-	-	-	-	-	902757
	GEM	893162	13603	5243	2231	1074	688	555	375	323	272	917526
	BWA-Short	834519	10008	3535	1226	408	160	83	52	43	26	850060
	Seqalto	885208	5692	1712	723	306	164	102	63	33	32	894035
100bp	BatAlign	924272	7599	941	728	851	332	182	108	101	63	935177
	Bowtie2	866310	8685	2833	948	254	110	69	42	23	5	879279
	BWA-SW	794661	7	0	0	0	0	0	0	0	0	794668
	Stampy	913874	-	-	-	-	-	-	-	-	-	913874
	GEM	875333	10327	3533	1445	638	377	283	193	163	178	892470
	BWA-Short	484558	5207	1747	662	211	112	79	48	20	13	492657
	Seqalto	890336	1821	515	194	91	44	38	11	3	8	893061

Table 7.1: *The results of mapping simulated datasets of lengths 75bp, 100bp and 250bp and reporting the top 10 hits. The correct hits are broken down by the rank of the hit. For 75bp and 100bp reads, BatAlign produces the most number of correct hits within its top 10 hits. For 250bp BatAlign misses only a small percentage of hits.*

reported 10 hits on both datasets. In fact, BatAlign recovers more hits as the top hit than all the other aligners do even after considering their top 10 hits. We attribute this to the effectiveness of the reverse alignment strategy and deep-scanning. This will be discussed more in Section 7.19.

7.17.2 Evaluation on Simulated Pure-Indel Reads

The reads generated by ART have only a ∼0.01% probability of containing an indel, and the previous statistics do not properly demonstrate each programs ability to identify indels. Therefore, we used the pure-indel read class to investigate the capabilities of each aligner to identify indels of varying lengths. To get the overall picture of sensitivity vs. accuracy we used the F-measure. (The sensitivity (SEN) and accuracy (ACC) are given by the equations $SEN = TP/(TP + FN)$ and $ACC = TP/(TP + FP)$, where TP and FP are the number of true and false positives respectively. Then we have

F-measure=$2(SEN * ACC)/(SEN + ACC)$.)

Figure 7.3 plots the accuracy rates and F-measure graphs across all the pure-indel datasets. BatAlign, Bowtie2 and Stampy had similar sensitivity across all the datasets of varying indel sizes, while the other programs generally have problems maintaining sensitivity when the size of the indel increases. We also observed that BatAlign had the smallest drop of sensitivity from 92.0% to 91.4% when the insert size was increased. In terms of accuracy, BatAlign has an average of 6.2%, 0.4%, 5.7%, 8.3%, 0.8% and 7.44% more specificity compared with Bowtie2, BWA-SW, Stampy, GEM, BWA-Short and SeqAlto respectively. Figure 7.3(A-B) also show that BatAlign maintained a high specificity even when the indel size increased. In terms of the F-measure, BatAlign clearly outperformed other methods and had the highest F-measure. It was the only program to have a stable F-measure of 95.5% across all indel datasets.

7.18 Mapping Real-Life Data

We downloaded 500k reads of 2 x 76 bp (DRA accession DRR000614, Sample: NA18943), 2 x 101 bp (SRA accession SRR315803, Sample: NGCII082 Mononuclear blood) and 2 x 150 bp (SRA accession ERR057562, Sample:ERS054071) paired-end datasets. The sequencing platform used for the downloaded data was Illumina Genome Analyzer IIx for the 76/101 bp datasets and Illumina MiSeq for the 150 bp dataset. In order to address the lack of a ground truth, we mapped the paired-end reads as single-end reads and calculated the fraction of reads that were mapped concordantly. We consider a pair of reads to be concordant if they have the correct orientation and maps within 1,000 bp of each other with a non-zero mapQ. (The distance 1000 is chosen since Illumina machines normally cannot extract paired-end reads from DNA fragments of size longer than 1000bp.) If both ends of the paired-end reads were mapped but were not concordant, they were marked as discordant. This form of verification [74] gauges the single-end mapping algorithms with reads containing the true spectrum of polymorphisms.

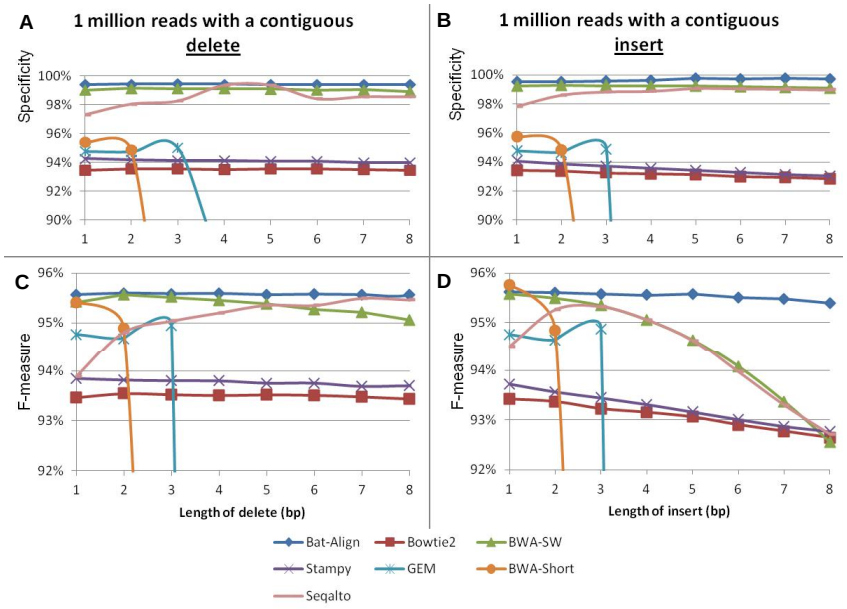
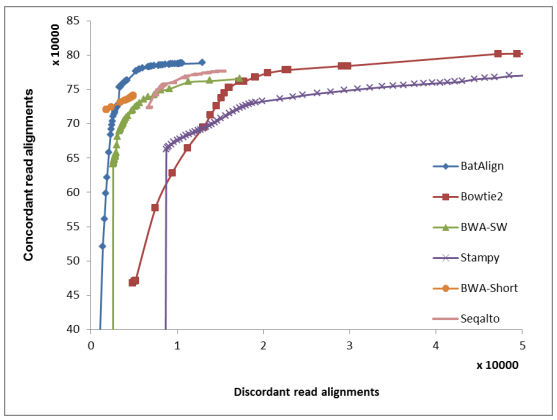
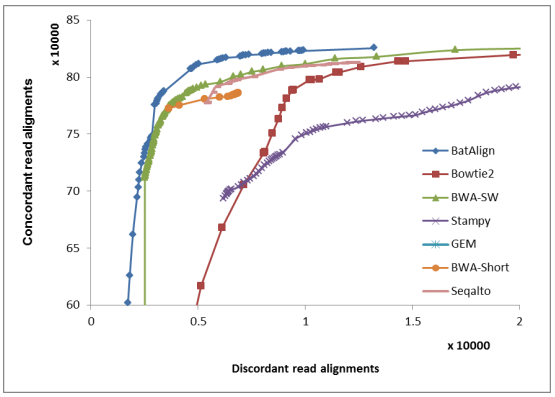


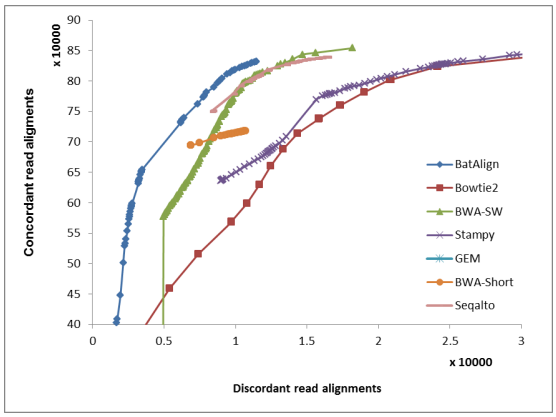
Figure 7.3: *Comparison of aligners capabilities in detecting indels in pure-indel datasets. The indel datasets were constructed by introducing indels of different lengths to a million reads simulated from hg19 that were error free. Among the aligners BatAlign shows the highest specificity and the best F-measure. It is also robust in detecting indels of different lengths, as can be seen by its stability of the F-measure.*



(a) ROC for 76 bp



(b) ROC for 101bp



(c) ROC for 150 bp

Figure 7.4: *Mapping of real-life datasets containing reads of length 75bp, 101bp and 150bp. One side of paired-end datasets were mapped and if the mate of a read was mapped within 1000bp with the correct orientation, the read was marked as concordant.*

Figure 7.4 shows the ROC drawn with the number of concordant and discordant reads and using mapping quality as the threshold. Consistent with the simulated data, BatAlign reported more concordant mappings and less discordant mappings compared to other methods over a large range of mapping quality cutoffs.

7.18.1 Running Time

BatAlign was developed focussing primarily on producing accurate alignments. From Table 7.2, we see that the default mode of BatAlign can run in a reasonable time

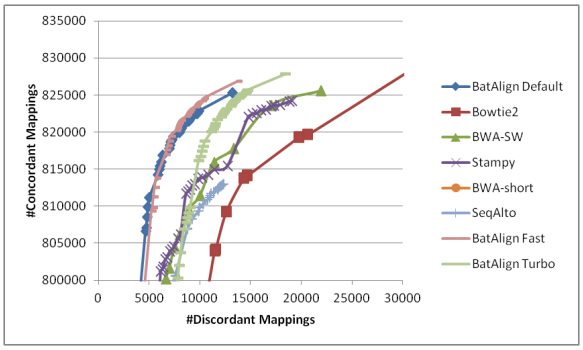


Figure 7.5: *ROC curves of BatAlign's fast modes compared with the ROC's of other aligners for a 100bp real life dataset. The faster modes of BatAlign still perform well compared to other aligners.*

compared to most of the fastest methods used in our comparisons. BatAlign trades speed for accuracy and was not as fast as GEM and Bowtie2. From the evaluation of simulated/real reads, the ROC curves show that BatAlign is more specific/concordant over a large range of mapping qualities. Since this is so, we try to allow more flexibility for the users by allowing BatAlign to run faster but with a slight decrease in accuracy.

The analysis of runtime was done with the same set of real data used in the previous section to present realistic timings of all the compared programs. Table 7.2 shows the runtimes and speedup factors between the programs, where Stampy was used as the baseline for measuring the speedup factor. GEM is the fastest aligner, and BatAlign in its turbo mode is the second fastest.

Figure 7.5 shows the ROC curves comparing different modes of BatAlign with other methods. We observed that the discordance rates increased with decreasing running times. This resulted from the reduction of search space in a bid for faster run times. Since BatAlign always tries to scan for all candidate hits incrementally and report the best hit, a reduction in search space will actually discard some concordant hits prematurely and cause an increase in discordance rates.

Program	Runtime(seconds)	Speedup factor
BatAlign - Default	583	6.3x
BatAlign - Fast	481	7.7x
BatAlign - Turbo	331	11.2x
GEM	214	17.3x
Bowtie2	459	8.0x
BWA-Short	598	6.2x
BWA-SW	639	5.8x
SeqAlto	677	5.5x
Stampy	3694	1.0x

Table 7.2: *The timing for mapping real-life data set of length 101bp. The baseline for speed comparison is taken to be Stampy, and the speedup of other methods compared to it are given. The fastest timing is reported by GEM. BatAlign in its default mode is slower than Bowtie, but faster than BWA aligners. In its faster modes, BatAlign is faster than or has a similar timing to Bowtie2.*

7.19 Discussion

Seeding methods are commonly used by aligners when aligning long reads due to the higher prevalence of structural variations and indels in them. Usually the seed size used is around 20-30 bases, and not many mismatches are allowed. Although these seeds produce many potential hits, when the seeds are extended many of them turn out to have a large edit distance from the original read. However, there is no proper way to filter out these type of seeds without actually doing read extensions. We propose to use larger seeds of around 75 bases, allowing a reasonable number of mismatches and indels appropriate for such a length. The longer anchors will produce a smaller set of seeds. Furthermore our reverse alignment strategy only inspects the most probable of these seeds. The advantage of this strategy can be seen from table 7.1. The total number of top 10 hits produced by BatIndel is less than the other aligners. However, it contains the most number of correct hits, indicating the ability of reverse alignment to produce a set of compact and accurate seeds.

Another reason contributing to the accuracy of BatAlign is the deep-scanning. To illustrate its effect, we mapped a million ART simulated 75bp reads with BatMis and BatAlign. BatMis scans incrementally for mismatches, and reports the unique hits without performing a deep-scan. Apart from indel scan and deep-scan, BatMis's mapping strategy is quite similar to that of BatAlign. Out of the million reads, BatMis mapped 917789 hits correctly and 4,625 wrongly. For BatAlign, 894773 hits were mapped correctly and 19 hits wrongly (We imposed a mapQ cutoff of 30 as a uniqueness condition for BatAlign hits).

The mapping results show BatAlign to be an accurate aligner. Simulated data shows that it is extremely robust when handling both insertions and deletions. Overall, its speed is comparable to the fast aligners available today. Therefore, we can conclude that BatAlign will be a useful option when mapping NGS reads.

7.20 Acknowledgement

This project is being carried out under the supervison of Wing-Kin Sung, including me and Jing Quan Lim. I would like to thank Jing Quan Lim for carrying out the experiments in this section.

Chapter 8

RNA-seq Alignment

8.1 Introduction

RNA-seq mapping is one of the most difficult problems in sequence mapping. Theoretically, it can be considered as mapping a read allowing multiple long gaps along with mismatches. The mapped portions of the read would correspond to exonic regions and gaps would correspond to intronic regions. However, for a genome like the human genome, the smallest exon size can be as small as 2 bases while the intronic gaps can span larger than 490,000 bases [124]. Therefore, practical solutions have to limit the sizes of exons and introns allowed in a read.

Another problem faced in RNA-seq mapping is the presence of pseudogenes. Like repeat regions for genomic mapping, these are a source of false mappings. We can categorize RNA-seq reads as exonic and junction reads; If the whole read originates completely within an exon we call it an exonic read. Otherwise, the read crosses one or more splice junctions and is categorised as a junction read. If an aligner is very passive towards filtering out pseudogenes, it will misalign junction reads. On the other hand if an aligner is too aggressive towards junction finding, it will missalign exonic reads.

In this chapter, we introduce BatRNA. BatRNA uses the BatAlign algorithm and

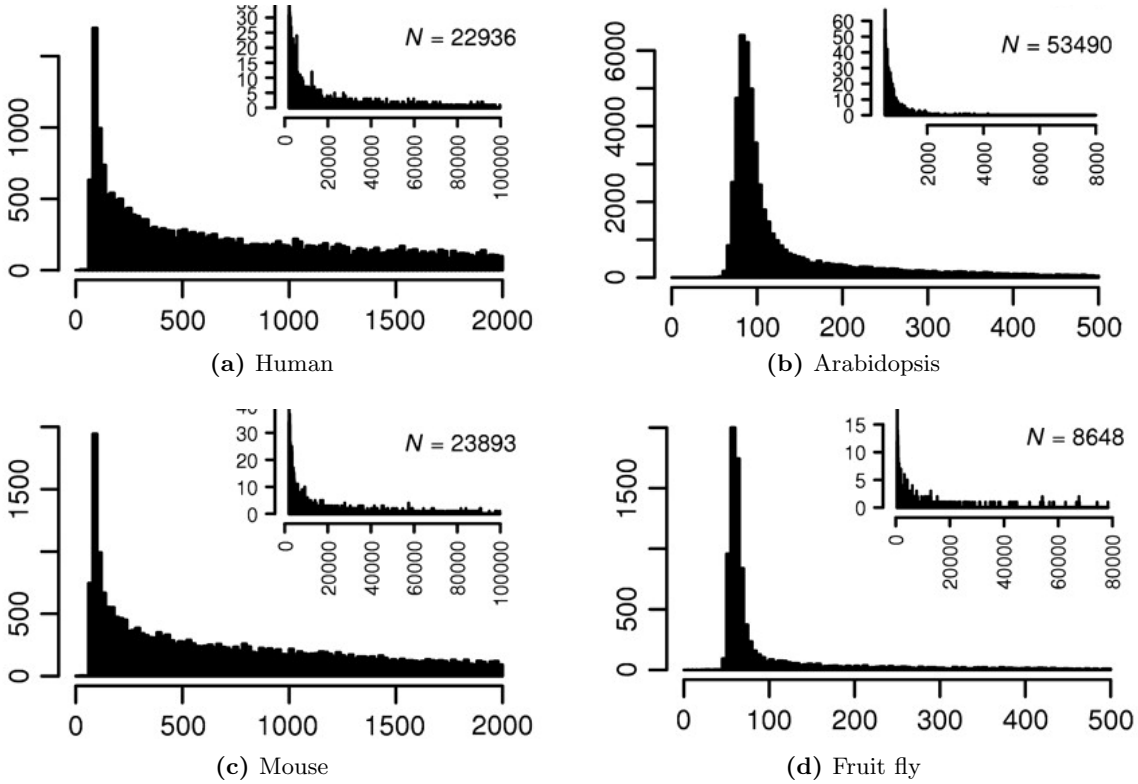
efficient splice junction detection algorithms based on the data structure described in section 3.7. BatRNA is a fast aligner and is the fastest among the compared methods that use a similar amount of memory. In addition, it can correctly resolve junctions containing only a small part of their constituent exons. BatRNA can strike a good balance between mapping junction and exonic reads; BatAlign is used to align exonic reads accurately and the splice junction detection algorithms can identify junctions that occur commonly.

The algorithm has two passes. The first pass identifies a set of confident junctions. This set of junctions is used in the second pass to resolve junctions from reads having only a short overhang aligning with a constituent exon, and to correct junctions that were misaligned by BatAlign as exonic reads. This two-pronged approach results in better mapping than algorithms that rely solely on split mapping or transcriptome methods.

8.2 An Alignment Score for Junctions

Suppose a read R contains a single junction consisting of two parts $R_1 = R[1, r]$ and $R_2 = R[r + 1, |R|]$. These will correspond to two exonic segments in the reference genome $T_1 = T[x, x + r - 1]$ and $T_2 = T[y, y + |R| - r - 1]$ respectively. We can denote this splice junction using the 4-tuple (R, x, y, r). We will now propose a method to assign a score to such a junction alignment. One obvious score is the Hamming distance $S_{Ham}(R, x, y, r) = H(R_1, T_1) + H(R_2, T_2)$ between the segments. A larger Hamming distance implies a weak alignment.

Most splice junctions exhibit strong and conserved motifs near acceptor and donor sites. A recent study [67] show that out of 53,295 confirmed junctions, 98.12% sites have the canonical GTAG signal. Next, semi-canonical junctions occur very rarely with 0.76% sites having the GCAG signal and 0.10% the ATAC signal. The rest of the junctions show 177 types of diverse signals. We will assign a score $S_{type}(R, x, y, r)$ to



(a) Human **(b)** Arabidopsis

(c) Mouse **(d)** Fruit fly

Figure 8.1: *Intron size distributions in human, mouse, Arabidopsis thaliana and fruit fly genomes. The inset histograms continue the right tail of the main histograms. For the histograms, bin sizes of 5 bp are used for Arabidopsis and fruit fly, while 20 bp bins are used for human and mouse. Source [47]*

the junction (R, x, y, r) by giving the highest score to the canonical junctions (default 3), second highest to the semi-canonical junctions (default 2) and the least score to any other junctions (default 0); i.e, a higher value of S_{type} indicates a more confident junction.

The next score S_{intron} takes in to account the size of the intron contained in the splice junction. Figure 8.1 shows the distribution of intron sizes for four different species. They all show a sharp peak at around 100 bp. The distribution of intron size is very tight for the relatively compact genomes of arabidopsis and fruit fly, but the human and mouse have a long tail. Very low and very high intron lengths are rare. A score can

be assigned based on intron length distribution so that intron sizes with most number of introns are penalized less. Based on the general shape of the intron size distribution, a straightforward scoring method is to use a logarithmic score so that longer introns get a lower score [30]. On the other hand we can attempt to model the exact intron distribution [43], but that kind of precision is not really necessary. Instead, we partition the possible intron lengths into four ranges based on [67]. Let K be a near the peaks in the intron size distribution of the reference genome. The intron sizes falling into intervals [K,2kb),[2kb,5kb),[5kb,30kb),[0,K) and [30kb,∞] will be assigned scores in decreasing order. If we set $K = 100$, the first two intervals will contain about 55% and 21% of human introns respectively, while the final interval will contain only less than 5% of all human introns. This partitioning is generally good for most species.

The final alignment score for a junction AS_{junc} is given by $AS_{junc}(R, x, y, r) = S_{type}(R, x, y, r) + S_{intron}(R, x, y, r) - S_{Ham}(R, x, y, r)$. A higher junction score implies a higher confidence in the junction alignment.

8.3 Basic Junction Finding Algorithm

Let R be a read containing a single junction, and assume that at least l bases from the constituent exons of the junction are present in R. If the maximum intron size we allow is D, then the l-mer prefix and suffix of R must be at most $D + |R| - 2l$ bases apart. If we assume that they contain no mismatches, to resolve the junction in R we can first find all the pairings of the l-mer prefix and suffix that are a distance $D + |R| - 2l$ apart using the methods described in section 7.3. Then the alignment scores of possible junctions occurring between these pairings can be computed. Finally, the highest scoring junction can be reported as the most likely junction.

Algorithm 8.1 describes a procedure *Find_Single_Junc* to enumerate the highest scoring junctions in the read following this idea. The algorithm has two steps. Step 1 (Line 1) finds all pairings of the l-mer prefix and suffix of R within a valid distance to

contain a junction. If a junction exists, it must lie in $R[l+1, |R|-l]$. Step 2 calculates the alignment score for all the possible junctions of all the pairings found (Lines 3-5). If (x, y) is a pairing, it would mean that the alignment score would be caculated for all the junctions $(R, x, y-(|R|-r-l), i)$ for $i = l+1, \ldots, |R|-l$. Finally, the best junctions are reported (Lines 6-8, 12).

Algorithm 8.1: $Find\_Single\_Junc(R, T, D)$.

Input: a read R, a reference T, a maximum intron length D
Output: Junctions with the optimum alignment scores

```
1  X= Pairings of R[1, l-1] and [|R|-l, |R|] within a distance D+|R|-2l apart ;
2  Max_Score = -∞; Best_Junc = ∅;
3  foreach pairing (x, y) in X do
4      for each junction between (x, y) do
5          S = Alignment score of the junction;
6          if Max_Score ≤ S then
7              if Max_Score < S then Best_Junc = ∅;
8              Add junction to Best_Junc;
9          end
10     end
11 end
12 return Best_Junc;
```

8.4 Finding Multiple Junctions

Consider a reference genome T and a read R. We now aim to align the read R onto the genome T allowing multiple junctions. Precisely, we partition R into segments $R_0, \ldots, R_p$ and align R_j without gaps to $T_j = T[i_j, i_j + |R_j| - 1]$, for $j = 0, \ldots, p$, satisfying the following conditions.

1. For each segment R_j where $0 < j < p$, $|R_j| \geq \lambda > 0$.
2. The distance between segments T_j and T_{j+1} given by $d_j = i_{j+1} - i_j - |R_j|$ satisfies $0 < D_1 < d_j < D_2$.

3. In any splice junction inside R, the $l(< \lambda)$ bases to the left and the right of the junction are present in R.

The T_j's and R_j's correspond to exons and the condition 1 limits the minimum size of an exon allowed inside R. The d_j are the intron sizes, and D_1 and D_2 are the smallest and largest intron sizes allowed.

The basic idea of finding multiple junctions is to find the junctions in the read R from left to right one by one, examining l bases at a time. The algorithm is based on the following lemma.

Lemma 8.4.1. *Consider a read R which aligns to the genome T according to the above criteria. For any position a, there are three possible cases:*

1. *There are no junctions in $R[a, a + l - 1]$.*
2. *There is exactly one junction in $R[a, a + l - 1]$, and $R[a + l, \min(a + 2l - 1, |R|)]$ has no junction.*
3. *There are two junctions, the first of which is in $R[a, a + l - 1]$ and the second is in $R[a + l, \min(a + 2l - 1, |R|)]$.*

Proof. For any other possibility, we would need to have more than one junction inside $R[a, a + l - 1]$ or $R[a + l, \min(a + 2l - 1, |R|)]$. Then, at least one exon would be completely contained in one of these segments. Since we have the restriction of the minimum size of an exon being $\lambda > l$, this would be impossible. Note that if $\lambda > 2l$, case 3 is not possible. □

We will also need the following lemma for Algorithm 8.2.

Lemma 8.4.2. *Suppose there is a single junction in $R[a, b]$, where $a < b < a + \lambda$. Then $[b + 1, a + \lambda - 1]$, is part of an exon to the right of the junction.*

Proof. If the junction starts at a, then $R[a, a+\lambda-1]$ is part of an exon. If the junction occurs at b, $R[b, \min(b+\lambda-1, |R|)]$ will be a part of an exon to the right of the junction. Therefore, their intersection given in the lemma is part of an exon for any junction occurring between $R[a, b]$. □

Corollary 8.4.3. *For $b = a + (\lambda - l) - 1$, $[b+1, b+l]$ is part of an exon of length l to the right of the junction.*

Proof. Follows trivially from Lemma 8.4.2 by substituting $b = a + (\lambda - l) - 1$. □

8.5 Algorithm for Multiple Junctions

We will describe an algorithm that can partition a read R into junctions consistent with the conditions set in the previous section. The process starts by assuming that the leftmost l bases of R lies completely inside an exon. This should be true if R satisfies condition 3. Then we move from left to right l bases at a time, either extending the current exon or finding junctions within the l bases. If we find a junction at the x^{th} base, we report it and start this process fresh, this time starting at the $(x+l)^{\text{th}}$ base of R.

Suppose we are currently at the a^{th} base of R, and $R[a-l]$ is part of an exon. Then all the junctions that can be found in the l bases $R[a, a+l-1]$ can be found as follows. If case 2 is true we can find all the possible junctions in $R[a-l, a+2l-1]$ using Algorithm 8.1. If case 3 is true, the junctions start somewhere in $R[a, i]$, where $i = 1, 1, \ldots, l-1$, and will extend at least l bases according to condition 3. We can find all these junctions by using Algorithm 8.1 on $R[a-l, a+i+l-1]$ for $i = 0, 1, \ldots, l-1$.

Based on this idea, the procedure *Seek_Junc* described in Algorithm 8.2 reports a set of putative transcripts for a read R. Each transcript is represented as a set of m junctions (S_i, p_i, q_i, r_i), where $i = 1, 2, \ldots, m$. *Seek_Junc* is a recursive algorithm that searches l bases of R at each recursion for a junction or exonic region from left to right.

$Seek\_Junc$ takes a suffix S of R, and three numbers a, L and j as arguments. $Trans$ is an array containing a partial transcript consisting of junctions found before entering the current iteration of $Seek\_Junc$. $S[1, a]$ will be part of a putative exon. L is the number of putative junctions in $Trans$; i.e. $Trans[1], \ldots, Trans[L]$ will contain the junctions 1 to L that were discovered in $R[1, a-1]$ in the order they were found. If $L \neq 0$, $T[j, j+a-1]$ will be the exonic region corresponding to $S[1, a]$.

The routine will first check if there are less than l bases to be processed (Line 2). If this is the case, and a junction has already been found ($L > 0$) we try to match the remainder of S with the reference (Line 3). The match will be determined based on the number of mismatches and/or the quality of mismatches. If a match is confirmed, then the transcript $Trans$ is reported (Line 4).

If there are more than l bases left, a junctions search and exon extension is performed based on the three cases given in Lemma 8.4.1. We first try to extend a known junction or the a-mer prefix of S right by l bases (Line 12). A successful extension is determined based on the number of mismatches and/or the quality of mismatches. If the extension is successful, the string $S[a+l, |S|]$ is recursively examined with $Seek\_Junc$ (Line 13).

Next we consider the case 2 of Lemma 8.4.1, where junctions between $S[a, a+l-1]$ will be added. We do this by using $Find\_Single\_Junc$ on $S[a-l, \min(a+2l, |S|)]$. If (S, p, q, r) is such a junction, it needs to be further validated (Line 17). For the validation, we match $S[1, a+r]$ and $T[p-r-a, p]$ taking into account the number and/or the quality of mismatches. Furthermore, if $j \neq 0$, then the condition $p = j + r$ must be satisfied so that it is consistent with the last junction found and $T[j, p]$ will form a contiguous exon. Finally, the junction must satisfy the constraints of the intron sizes. If the junction is valid, it will be added to the current partial transcript (Line 18). Then, S is trimmed near the junction, and $Seek\_Junc$ is called recursively to search for the remaining junctions on the trimmed read (Line 19).

Finally, a search is made for junctions due to small exons given by case 3 of

Algorithm 8.2: Seek_Junc$(S, a, L, j, Trans)$

Input: $S[a, |S|]$ is the part of the read being searched for junctions, L is the number of junctions stored in putative transcript $Trans$, $T[j, j + a - 1]$ is corresponding to putative exon $S[1, a]$ if $j \neq 0$.

Output: Report possible transcripts

```
1  if L > 0 then
2      if a > |S| − l then
3          if S[a + 1, |S|] and T[j + a, j + |S| − 1] match then
4              Report Trans;
5          end
6      else
7          if S[a + 1, a + l] and T[j + a, j + a + l − 1] match then
8              Seek_Junc(S, a + l, L, j, Trans);
9          end
10     end
11 else
       /* -------------------- Extend exon ----------------------    */
12     if S[1, a] can be extended l bases then
13         Seek_Junc(S, a + l, L, j, Trans);
14     end
15 end
16 if a > l then
       /* ------------------Single Junction----------------------    */
17     foreach each valid junction (S, p, q, r) between S[a, a + l] do
18         add junction to Trans[L];
19         Seek_Junc(S[r + 1, |S|], 1, L + 1, q, Trans);
20     end
       /* --------------- Small exon if 2l > λ -------------------    */
21     for i = a to a + l − 1 step (λ − l) do
22         for each valid junction (S, p, q, r) between S[i, i + (λ − l)] do
23             add junction to Trans[L];
24             Seek_Junc(S[r + 1, |S|], 1, L + 1, q, Trans);
25         end
26     end
27 end
```

Lemma 8.4.1. When $\lambda > 2l$ this part of the algorithm is not necessary. Note that there are actually two junctions within $S[a, a + 2l - 1]$, but we only search for the leftmost junction. We only need to check the interval $S[a, a + l - 1]$ for junctions. A naïve search for these junctions can be done by using $Find\_Single\_Junc$ on the strings $S[a - l, a + i + l - 1]$ for $i = 0, 1, \ldots, l - 1$. However, according to Corollary 8.4.2, if there is a junction inside $[a + i(\lambda - l), a + (i + 1)(\lambda - l) - 1]$, then the l bases starting at $a + (i + 1)(\lambda - l)$ must be part of the exon to the right of the junction. Therefore, if we use $Find\_Single\_Junc$ with $S[a+i(\lambda-l)-l, a+(i+1)(\lambda-l)+l-1]$ for $i = 1, \ldots, \lfloor (l - 1)/(\lambda - l) \rfloor$ all the possible junctions can be found more efficiently(Lines 21,22). If junctions are found, similar to case 2 of Lemma 8.4.1, their consistencies are checked. Then S is trimmed at the junction and $Seek\_Junc$ called using the trimmed S (Line 24).

8.6 Details of Implementation

The most time consuming part in the $Seek\_Junc$ algorithm is the finding of small exon junctions corresponding to case 3 of Lemma 8.4.1. Furthermore, it generates a high number of false hits. We set $l = 18$ and assume that $\lambda > 2l$. This choice of $\lambda > 36$ is valid for more than 99% of exons. The number of mismatches allowed must also be set low for faster speeds. We allow only one mismatch in the l-mers used in $Find\_Single\_Junc$ by default. The extension of junctions is done so that at each extension, the total quality of mismatches do not exceed that of a single high quality mismatch.

Sometimes the algorithm fails to find junctions due to having too many mismatches near ends or due to the presence of residues of length less than l from some exon. Some of these junctions can be recovered by trimming the reads from the ends and using $Seek\_Junc$ on the trimmed reads. If a read fails to produce junctions, it can be further trimmed. The trimming is applied 5 bases at a time until junctions are found or until

the trimmed read size becomes less than 50 bases.

8.7 Mapping with BatAlign

For the human genome, the average length of an exon is around 235bp and about 33-38% of the reads would contain more than one junction in a 100bp read [59]. This would suggest that most of the reads would either contain a single junction or no junction at all. In either case, most of the reads would contain a sizeable portion (for 100bp reads larger than 50 bases) from an exon. Although BatAlign is basically designed to map reads allowing mismatches and a single short indel, the previous statistic suggests that BatAlign can be useful in two ways as a pre-mapper for RNA-seq reads.

The first use of BatAlign is to map exonic reads. This case is exactly similar to mapping normal genomic reads, and with BatAlign's high accuracy, we can expect exonic reads to be aligned quite reliably. However, we have to consider the possibility of a junction read mapping to a pseudogene during this step. If the alignment is weak we will consider the read for junction finding.

The second use of BatAlign is to find an anchor for one side of a junction in junction reads. BatAlign can perform local alignment, so if there is a sufficiently long portion of an exon, BatAlign may recover it. We can inspect the alignment given by BatAlign and choose the largest contiguous region in the alignment as a potential exonic region. Using this as an anchor point, we can examine its neighbourhood for a potential junctions.

The procedure *BatAlign_Junc* recovers junctions based on this idea and is given in Algorithm 8.3. The procedure is passed the largest contiguous segment in the alignment $R[x, y]$ and the corresponding substring in the reference genome $T[x', y']$. For example, if the alignment is given by the CIGAR string "`5M6D6M1I25M`", the largest matching segment will correspond to the alignment `25M`, and it is assumed that this portion is from the exon to the right of the junction. *BatAlign_Junc* first checks to see if there is a sufficiently large anchor of Min bases from the read that can potentially be from

the left side of the junction (Line 4). Then it will search for all the occurrences of this anchor in a neighbourhood D bases (with default value of $D = 20,000$) away from the right anchor $T[x', y']$ (Line 5). Next, the alignment scores of all the junctions $(R, x', q - |R| + r + 1, r)$ (where $r = 1, \ldots, x - 1$) between the right and left anchors are calculated (Lines 7-8). The best scoring junctions are stored in $Juncs$ (Lines 9-11). Similarly, Lines 16-27 will try to find a junction to the right of the largest matching segment by finding a right anchor.

8.8 BatRNA Algorithm

BatRNA algorithm puts together the basic algorithms given in the previous sections. BatRNA consists of two passes through the reads. In the first pass, a reliable set of exons and junctions will be found, and a set of reads that possibly contain junctions are separated. In the second pass, these separated hits are checked to see if they contain a junction found in the first pass.

The first pass starts by mapping all the reads with BatAlign. If a read maps without indels and mismatches, it is assumed to be a correct exonic read and is reported. Otherwise, $Seek\_Junc$ is run to discover potential transcripts. If a potential transcript cannot be found, reads are trimmed at the ends as described in Section 8.6 and $Seek\_Junc$ is run on the trimmed reads until a potential transcript is not found. If there is no potential transcript even after this stage the alignment produced by BatAlign is examined. If it contains soft clippings or more than two indels, $BatAlign\_Junc$ is used to find a junction using the largest contiguous part of the alignment as an anchor.

At the end of this first pass, if a read produces potential transcripts as well as an alignment by BatAlign, we determine which one is the better one. The potential transcripts are assigned a score $S'_{Mis} - S'_{type} + S'_{Gap\_Size} + S'_{Trim}$, where S'_{Mis} is the number of mismatches in the potential transcript alignment, S'_{type} is a score based on the presence of canonical/semi-canonical motifs at the splice junctions, S'_{Gap_Size} is a

Algorithm 8.3: $BatAlign\_Junc(R, T, D, Min, (x, y), (x', y'))$.

Input: a segment $R[x, y]$ that aligns with the substring $T[x', y']$ of T, a maximum intron length D, and a minimum anchor size Min
Output: Find Junctions in the neighbourhood of $T[x', y']$

```
1  R[x,y]= Largest contiguous portion in the alignment of R;
2  T[x',y']= Region in T that aligns with R[x,y];
3  Max_Score = −∞;Juncs = ∅;
   /* Find junctions to the left                                    */
4  if x > Min then
5      Res = occurrences of R[1, Min] in T[x' − D, y'];
6      foreach T[p,q] in Res do
7          foreach junction in T[q,x'] do
8              Score = alignment score of the junction;
9              if Score >= Max_Score then
10                 if Score > Max_Score then Juncs = ∅;
11                 Max_Score = Score; Add junction to Junc;
12             end
13         end
14     end
15 end
   /* Find junctions to the right                                  */
16 if y < |R| − Min then
17     Res = occurrences of R[|R| − Min + 1, |R|] in T[y', y' + D − 1];
18     foreach T[p,q] in Res do
19         foreach junction in T[y',p] do
20             Score = Alignment score of the junction;
21             if Score >= Max_Score then
22                 if Score > Max_Score then Juncs = ∅;
23                 Max_Score = Score; Add junction to Junc;
24             end
25         end
26     end
27 end
```

score based on the intron sizes of the junctions and finally S'_{Trim} a penalty based on the number of bases trimmed from the read. The least scoring potential transcripts are chosen as the best potential transcripts. If the score of the best potential transcripts is less than the total number of indels and mismatches in the BatAlign alignment, and there is a unique best potential transcript, it is reported. If the score is smaller the BatAlign alignment is passed to the second stage. Otherwise, both the best potential transcript and the BatAlign hit is output with a mapping quality 0.

8.8.1 Realignment of Reads

The purpose of the second stage of the alignment is to correct misalignments by BatAlign. One reason for such misalignments is that sometimes $Seek\_Junc$ and $BatAlign\_Junc$ fail to align a read having just a few bases from a constituent junction. To correctly align these reads, we first see if there are any junctions from the first pass that overlap the BatAlign mapping. If there is such a junction, we try to re-align the read to this junction. If the re-alignment is successful, the junction is reported. Otherwise, the BatAlign alignment is reported.

Procedure $Realign\_Junc\_Left$ given in Algorithm 8.4 is based on this idea and shows how to re-align a BatAlign mapping that maps part of the exon in the left side of a junction correctly. $Juncs$ will contain the high confident junctions discovered in the first pass. A junction in $Junc$ will overlap with the BatAlign mapping if the distance between the donor site and the BatAlign mapping is smaller than $|R|$. Each of the junctions in $Junc$ will be examined to see if the left hand side of one of these junctions will overlap with the BatAlign mapping (Lines 2,3). Let $\delta = |R| - (p + r - x)$ be the length of the segment of R that does not overlap with the left side of the junction. If there is an overlap, $T_1 = [x, p + r - 1]$ should overlap $R_1 = [1, |R| - \delta]$ and $T_2 = [q, q + \delta - 1]$ should overlap $R_2 = [|R| - \delta + 1, |R|]$ (Lines 4-6). For a confident overlap, we will assume that the number of mismatches in the overlap does not exceed k, and store the confident

Algorithm 8.4: Realign_Junc_Left$(R, T, (x, y), Juncs)$.

Input: a set of junctions $Juncs$, a read R that that BatAlign maps to $T[x, y]$
Output: Re-alignment of R to a junction in $Juncs$

```
1  Hits = ∅;
2  foreach J = (R, p, q, r) in Juncs do
3      if p + r > x and p + r − x < |R| then
4          δ = |R| − (p + r − x);
5          T1 = [x, p + r − 1]; T2 = [q, q + δ − 1];
6          R1 = [1, |R| − δ]; R2 = [|R| − δ + 1, |R|];
7          if H(T1, R1) + H(T2, R2) < k then
8              add (R, x, p, |R| − δ) to Hits;
9          end
10     end
11 end
12 if |Hits| = 1 then
13     Report Hits;
14 end
```

junctions in $Hits$ (Lines 7,8). If there is only one such possible match it is reported (Lines 12-13). A similar algorithm *Realign_Junc_Right* can be constructed to re-align junction hits whose right hand side is correctly mapped by BatAlign.

8.9 Results

To demonstrate the performance of BatRNA with existing methods, we chose a set of state-of-the art aligners; OLego version 1.1.1, MapSplice version 2.1.2, STAR version 2.3.0e and TopHat2 version 2.0.8b. All the experiments were done on a system with an Intel Xeon X5680 processor and 144GB of RAM. 20 threads were used in all the mappings. We will asses the performance of each aligner on a set of simulated data and on a set of real life data.

The simulated datasets were generated by the BEERS package distributed with RUM [41]. The master configuration files used for the simulation were downloaded from RUMs website and are based on the hg19 RefSeq database. Each dataset consists of 2

million RNA-seq reads. Datasets of 76 bp and 100 bp read lengths were generated for our experiments. Additionally, BEERS was configured not to simulate reads coming from novel splice events by using the option `palt 0`. For testing the performance of real life data, we used the RNA-seq dataset ERP001196. This 90bp data set was generated from an Illumina HiSeq 2000 machine in a transcriptome study of humans [134].

8.10 Accuracy and Sensitivity in Simulated Data

We assessed the sensitivity and accuracy (For definition see Section 7.17.2) of each method using simulated reads for which their correct alignments were known. For each method and each dataset, we recorded the number of correct and wrong alignments. As mappings with zero mapQ will not be used by most downstream analysers like Cufflinks [138], these mappings are discarded. We then further categorise the reads in to exonic and junction reads. Due to the inability in aligning some indels back to their exact locations and the presence of soft-clippings, an alignment will be considered correct if the leftmost position falls within 50 bp of the Oracles position or if at least one intron gap was correctly identified.

		Junctions			**Exons**			
		correct	wrong	%correct	correct	wrong	%correct	F-measure
100bp	BatRNA	**374496**	**20555**	**94.80**	1506146	**3628**	**99.76**	**96.32**%
	MapSplice	**378214**	27837	93.14	**1521271**	26967	98.26	**96.07**%
	STAR	351931	50168	87.52	**1522616**	22366	98.55	94.98%
	Olego	324820	20915	93.95	1496843	**2445**	**99.84**	94.75%
	TopHat2	344165	**19879**	**94.54**	1499603	47746	96.91	94.28%
76bp	BatRNA	**305421**	**23226**	**92.93**	1584764	**3855**	**99.76**	**96.51**%
	MapSplice	**307093**	30650	90.93	**1604210**	35266	97.85	**96.11**%
	STAR	277200	58975	82.46	**1602617**	28961	98.22	94.75%
	Olego	252820	25207	90.93	1575932	**2687**	**99.83**	94.84%
	TopHat2	268674	**19348**	**93.28**	1580547	25915	98.39	94.97%

Table 8.1: *Statistics for different aligners when a simulated dataset of 2000 000 reads were mapped. The best two statistics of each column are shown in bold letters.*

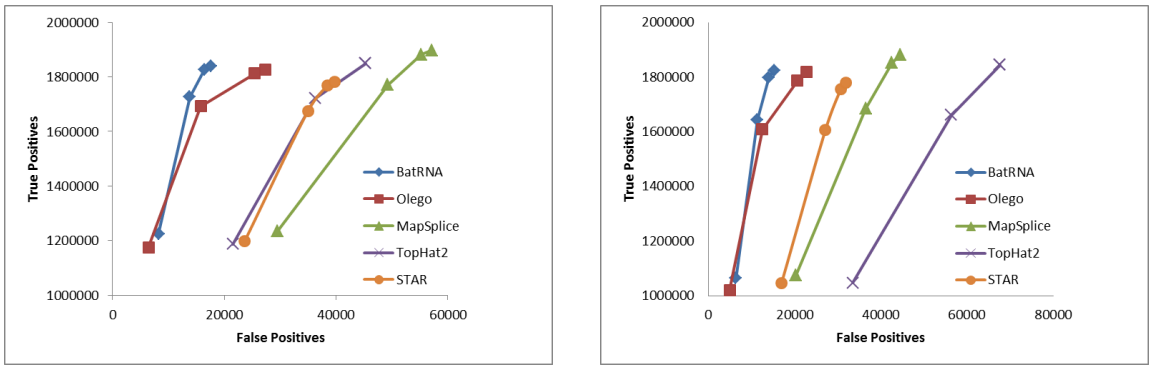
		1	2	3	4	5	6	7	8	9	Total
100bp	BatRNA	32.45	47.21	51.73	53.78	51.67	53.24	55.17	58.20	60.80	52.55
	Olego	30.39	28.93	24.08	16.96	9.85	5.05	2.11	47.82	50.94	26.84
	MapSplice	33.39	38.04	46.10	52.87	57.39	58.42	60.01	60.84	61.08	53.18
	STAR	33.16	32.95	32.66	32.81	33.57	33.33	33.57	32.74	35.54	34.18
	TopHat2	46.66	46.53	47.35	47.50	48.38	47.21	48.19	49.11	48.63	47.86
76bp	BatRNA	37.03	55.01	59.82	62.42	59.51	60.13	62.95	67.02	70.14	60.49
	Olego	35.10	34.42	28.49	20.60	11.14	5.49	2.53	55.42	61.41	31.73
	MapSplice	37.42	43.90	52.28	60.82	64.98	64.50	66.77	67.76	69.54	60.23
	STAR	37.79	38.50	37.91	38.03	38.32	36.93	37.99	38.18	41.01	39.19
	TopHat2	52.30	53.17	53.38	53.36	53.07	50.65	51.48	52.07	53.26	52.66

Table 8.2: *Table showing the total percentage of junctions having short residues of size 1bp-9bp that were recovered by each program. The final column gives the total percentage of junctions having less than 9 bases that were recovered.*

Table 8.1 shows the results of this experiment. When discovering junctions accurately, BatRNA shows the best performance in terms of sensitivity and accuracy. When exonic reads are mapped it is not the best in terms of sensitivity, but shows quite good accuracy. Overall, it retains the best F-measure (For definition see Section 7.17.2) when the whole mapping is considered. We have marked in bold the top two performing aligners for each statistics. It can be seen that BatAlign is always among the best two aligners for each category, except for the sensitivity of exons. This statistic shows that BatRNA is generally the best aligner both for mapping junction reads and exonic reads.

8.11 Mapping Junctions With Small Residues

When a sizeable portion of the exons of a junction are present in a read it is relatively easy to find junctions. However, when there is only a short residue (i.e a short fragment) from an exon, most aligners fail to align them. Table 8.2 shows the ability of each aligner to detect junctions with small residues ranging in size from 1bp-9 bp. For the smallest residue of 1bp, Tophat recovers the most junctions. For other very small residues around 2bp-4bp, BatRNA shows the best recovery rate. MapSplice shows the



(a) 76bp **(b)** 100bp

Figure 8.2: *Total number of correct hits plotted against the total number of wrong hits for 0-3 mismatch hits. Only the high quality hits with mapQ>0 were considered.*

best recovery for 5bp-7bp residues, but BatRNA catches up fast. In total, MapSplice and BatRNA recovers about the same percentage of short residues. It can be seen that except for 1bp residues, BatRNA and MapSplice are always among the top two.

8.12 Accuracy of High Confident Mappings

Next, we show the accuracy of mappings having a small number of mismatches. These are likely to be the most accurate mappings. In this analysis we remove those hits with mapQ 0. For aligners outputting multiple hits, only the top hit is retained. We plot the total correct number of mappings against the total wrong mappings thresholded by the number of mismatches. Figure 8.2 shows the resulting graph. The graph shows that generally, BatRNA produces high quality alignments.

8.13 Real-Life Mappings

We mapped a sample of 2 million real life reads to determine the performance of each aligner on real life data. We then isolated a reliable set of hits from this mapping by throwing away multiple hits and those having a mapQ 0. Since we do not know where

Aligner	Known Junctions	Known Exons	Time (s)
BatRNA	45352	657986	81
MapSplice	41523	657675	418
STAR	43928	657075	11
Olego	44872	657665	272
Tophat2	30142	649459	694

Table 8.3: *The results of validating the junctions and exonic mappings found by each aligner on a real dataset containing 2 000 000 reads. The validation was done against known exons and junctions in the Refseq.*

the reads originated from, we make this filtering very stringent to get the most accurate hits by considering only those hits having 0 mismatches. We next consider mappings containing junctions and exons separately. We validate these hits by comparing against known junctions and exons in the Refseq annotation. Table 8.3 shows the results. All the aligners recover about the same number of exons. However, BatRNA shows a better junction recovery. We determined the timings for the mappings. The timing for STAR does not take into account the time taken to load its index (around 30 seconds) compared to negligible time to load indexes by other programs. The timings show that STAR is extremely fast. The next best timing was given by BatRNA. BatRNA shows a considerable speed advantage over the other aligners except STAR.

Unlike in the case of DNA sequence alignment, we cannot reliably validate concordantly mapped reads based on a maximum insert size. This is because the intron size between two exons can be several thousand bases long, thus placing an actually concordant pair of reads very far apart in the genome after mapping. Therefore, we did not attempt this type of validation.

8.14 Discussion

RNA-seq mapping is a complex problem in mapping reads. A good RNA-seq mapper should be able to map both exonic and junction reads reliably. Our solution BatRNA

is capable of doing both, and can sensitively detect splice junction even if only a small residue of exonic regions are present. It is faster than all solutions except STAR. However, when compared with STAR's high memory usage of around 30 GB, BatRNA might be a better solution for smaller computer systems.

Mapping simulated reads show that BatAlign can map purely exonic reads with an accuracy close to 99.8%. However, it will misalign junction reads, or will recover only a local alignment of them. We use these facts to our advantage and use BatAlign as a pre-mapper. When BatAlign's output is used with the result of junction finding algorithms, the best alignments can be determined.

BatRNA uses fast FM-index based junction finding algorithms. The algorithms are designed with parameters to be sensitive to most small exons. Furthermore, a second pass is done over non-junction hits to see if these can be re-aligned to junctions that have already been found. The end result is an aligner that is sensitive at detecting junctions.

Overall, the experiments indicate that it is among the best in most categories of performance. Therefore, it offers an attractive alternative for many users.

8.15 Acknowledgement

This project is being carried out under the supervison of Wing-Kin Sung, including me and Jing Quan Lim. I would like to thank Jing Quan Lim for carrying out the experiments in this section.

Chapter 9

Conclusion

9.1 Introduction

In this thesis I address the problem of aligning NGS reads efficiently and accurately. Various data structures have been used to align NGS reads. However, the current trend of sequence mapping algorithms is to utilise BWT based techniques. This has resulted in the introduction of many light-weight and fast sequence aligners. I have introduced several algorithms to align different types of NGS data and a new data structure based on the BWT.

The first algorithm BatMis can efficiently perform k-mismatch mapping and is the fastest among the compared methods. The new data structure I introduced is designed to find occurrences of l-mer patterns in a reference efficiently. Algorithms can be designed using this data structure so that gapped alignments can be performed efficiently. Combining these new algorithms, I presented practical solutions to mapping DNA and RNA-seq reads.

9.2 BatMis

When I started my Ph.D., the length of the sequences available were around 36 bp. The aligners designed at that time were concentrating on aligning these reads with mismatches, which was quite a reasonable method for reads having such a short length. Furthermore, the use of BWT to bioinformatics applications was novel, and BWA-SW was the only known program to use it. In this backdrop, BatMis was designed to provide a fast solution to read alignment allowing only mismatches, with a memory constraint of 4GB which was the commonly available memory configuration for PC's at that time.

BatMis outperforms other aligners in terms of mapping reads allowing mismatches. However, a large percentage of reads can be mapped back allowing only a low number of mismatches, and in general the gain in mapping allowing a large number of mismatches is low. For example, for the 35 bp dataset SRR001115, we found that 79.6% reads can be mapped back at 4 mismatches. If we increase the number of mismatches to 5, only an additional 2.9% mappings were gained. The difference between the mappings discovered when using high mismatches and employing heuristics was not significant. Furthermore, the extra mapping we gain tend to be noisy due to high mismatch content.

Nevertheless, the exact solution of k-mismatch mapping for small reads does have its uses. In many pipelines (e.g. RNA-seq), a read is fragmented in to small chunks and all the k-mismatch alignments of each chunk are inspected for further downstream filtering. BatMis will be very good at handling this kind of tasks. Based on the benchmark results, BatMis would give the best results faster or at a similar speed compared to other aligners. For example, BatMis was used as the mapper for the ChIA-PET [73] pipeline. In this pipeline, the fragments of paired end reads are mapped allowing a single mismatch and the resulting mapping was used to find chromatin interactions. The bisulphite mapper BatMeth [82] also uses the BatMis algorithm at its aligning stage to produce multiple hits. The choice of BatMis as the aligner is a major factor in BatMeth being one of the fastest and accurate bisulphite aligners.

From the feedback from users, BatMis is being used for mapping of ancient DNA. The ability of BatMis to allow large number of mismatches is useful for mapping DNA to distant reference genomes. Also, the algorithm has been incorporated into pipelines that look for CRISPR off-target sequences, where sequences of around 23 bp are mapped back allowing up to four mismatches to the human genome.

As the length of NGS reads became longer, BatMis needed to incorporate indel handling abilities as well. Although early releases of BatMis were able to detect one mismatch by simulating an indel during the search process, I needed to develop a more versatile aligner that can handle longer reads and indels.

9.3 BatAlign

BatAlign is my solution to general DNA read mapping. The benchmark tests show that it can be run at speeds comparable to the Bowtie aligner with an accuracy comparable to the BWA aligner. This makes it an attractive choice for people seeking a good balance of speed and accuracy. Simulated data show that BatAlign is the most robust aligner when mapping reads containing long indels.

BatAlign uses BatMis for mismatch mapping. Novel algorithms using my new data structure are used to find indels. Furthermore, a new semi-global alignment method and an accelerated Smith-Waterman algorithm are used with a seed and extend approach to handle long reads.

While the introduction of faster and accurate algorithms is always good, current sequencing depth and read length along with clever downstream analysing packages (e.g. GATK [100], Dindel [5]) has helped close the gap between aligners when it comes to SNV discovery and small indel detection. However, our group is working on improving the detection of structural variations, on which the aligner design may make a noticeable difference.

Our group now has a version of BatAlign that can handle paired-end reads. Our

experiments on simulated reads indicate that many aligners either map discordant reads as concordant reads or vice versa with a bias. That is, aligners prefer to either form concordant reads or discordant reads. The first type of aligners will miss real structural variants, while the second type will produce false structural variants. We have observed that our paired-end version of BatAlign is among the best at correctly determining concordant and discordant reads. With this line of work, we believe that BatAlign will be the aligner of choice for detecting structural variants.

BatAlign is designed with aligning reads having similar profile to Illumina reads and a reference genome having a polymorphism rate similar to the human genome. BatAlign will not be ideal when the reference genome is highly polymorphic or when aligning reads to unfinished genomes. An aligner like Stampy would be more suitable for these. Nevertheless, BatAlign can be used effectively on cancer genomes.

9.4 Improving Results

BatAlign and BatMis were designed primarily to solve the mapping problem rather than the alignment problem. In fact, the alignment problem is best tackled by methods that take in to account the global mapping, and not by aligners that map individual reads. We now have variant calling software like Samtools [76] that re-align reads based on the nature of multiple reads mapping to a given region. GATK can post process an alignment file and output the corrected alignments. These tools can be used in conjunction with BatAlign to improve the final result to obtain the best alignment.

9.5 BatRNA

BatRNA uses BatAlign to map exonic reads and anchor possible junction reads. The reads are also processed for junctions using fast junction finding algorithms based on my novel data structure. BatRNA attempts to avoid pseudogenes by comparing

the junction mapping and the exonic mapping. The junction mapping algorithms are designed to be sensitive, and even those junctions having a small residue from an exon can be discovered. Although it is slower than the aligner STAR, its run time is much faster than other programs. However, STAR uses 30GB of memory which might pose a problem for some users. Overall, experiments show that it has a good accuracy and sensitivity among all the programs compared.

Currently, BatRNA is run on *de novo* mode. However, with many genomes being well annotated for genes, it is good to have the option of using these annotations as guides during the mapping. This will lead to an increase in both the sensitivity and accuracy. Another problem associated with RNA-seq mapping is the presence of pseudogenes. If the aligner is aggressive towards eliminating pseudogenes, exonic hits will be classified as junctions hits. On the other hand, an aligner that is more passive about junctions will map reads to pseudogenes. However, table 8.1 shows that BatRNA is not too aggressive on both cases. We can use a database of known pseudogenes to further reduce the number of false mappings in future.

Single-cell data analysis has recently started to become popular. A problem with single-cell data is the lower abundance of transcripts compared to multi-cell samples. Therefore, it is important to achieve a good sensitivity in these datasets. I will be trying to improve BatRNA to address this concern in future work.

9.6 BWT-based Algorithms

The BWT-based algorithms are fast at performing exact searches. However, current BWT-based methods are not effective when a large number of mismatches are present. For example, the alignment method of bowtie is quite good at handling one-mismatch mapping and the cases of two mismatches that appear in the same half of the read. However, the method is difficult to extend as the configuration of mismatches becomes more complex. Similarly, the pruning strategies used by BWA fail almost totally when

the mismatch number is high. Compared to these aligners, BatMis select a set of non-redundant seeds, and does an incremental mismatch scan. Unlike other aligners, it makes extensive use of failed alignments at the previous stage of mismatch scanning. Overall, this will give BatMis a narrower search space to work with.

The BWT-based data structure used by BatAlign and BatRNA speeds up indel and splice junction searches. While most aligners search for a seed and extend method to look for indels, BatAlign can use the data structure in many cases to locate a narrower set of regions containing possible indels. This lookup, together with a fast novel algorithm to screen indels gives an advantage to BatAlign. When splice junctions are searched, current algorithms generally try to splice together clusters of seed mappings or extend them. However, BatRNA can quickly locate and initial set of possible junctions using the new data structure, and systematically extend them. This extension can also make use of the novel data structure to locate nearby junctions.

Oveall, we can conclude that the Bat family of aligners gain a speed advantage over other aligners due to their algorithms being more integrated with the BWT data structure.

9.7 Criteria for Benchmarking

There is a large number of aligners currently available in the literature. Some of these aligners are no longer maintained (e.g. Maq), cannot work with current sequencing data (e.g. G-Mo.R-Se) or are too specialised (X-mate). While it is not practical to make an exhaustive test against all the available aligners, I have made comparisons with peer reviewed aligners that are reputed for speed (GEM,Bowtie2,STAR) and are reputed for accuracy (BWA,Stampy,MapSplice). The general observation is that Speed is gained at the sacrifice of accuracy in all classes of aligners.

For the simulation datasets, we used the ART simulator. It was the only available simulator that can simulate quality scores realistically. In the absence of a ground

truth, it is difficult to directly measure the correctness of real life data. In the case of DNA reads, a test measuring the concordance of paired reads was used as this test has now become a somewhat standard method of gauging the accuracy of an aligner. For RNA-seq reads, we use the well annotated RefSeq database for verification of known junctions and exons. This approach too is a standard benchmarking method.

9.8 Future Directions

Analysing reads having thousands of bases will be common in the near future. Although the algorithms given in this thesis can be extended to handle them, it may be possible to produce faster and better alignments for these types of reads using different data structures. Also, the dynamic programming algorithms will be extremely slow at lengths of this magnitude. I will continue to generalise the fast global alignment method described in this thesis to handle such cases.

While still being a luxury for the average computer lab, large genome centers are comfortable with allocating 30-100 GB memory to a user. Targeting these users, faster algorithms can be developed if data structures other than the BWT are considered. For example, one of the reasons for STAR being fast is its use of suffix array based methods.

Another idea is to find a data structure that can do inexact matches. With this kind of a data structure, a query for say 1 mismatch hits would immediately give all the possible one mismatch hits, or a set of hits that contain all the one mismatch hits. If such an index is available, we can speedup BatMis by doing fast multi-mismatch scans where, for example, both exact matches and one-mismatch hits can be found in just one pass of the algorithm.

Furthermore, there are other sequence mapping problems that involve extremely large databases. A database like the nt-database keeps growing year by year, and indexing it efficiently is an interesting question. We plan to focus our attention on applying the methods developed in this thesis to such large databases.

Bibliography

[1] Abouelhoda, M. I., Kurtz, S., and Ohlebusch, E. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[2] Adams, M. D., Soares, M. B., Kerlavage, A. R., Fields, C., and Venter, J. C. Rapid cDNA sequencing (expressed sequence tags) from a directionally cloned human infant brain cDNA library. *Nat Genet*, 4(4):373–380, Aug 1993. doi: 10.1038/ng0893-373.

[3] Adessi, C., Matton, G., Ayala, G., Turcatti, G., Mermod, J. J., et al. Solid phase DNA amplification: characterisation of primer attachment and amplification mechanisms. *Nucleic Acids Res*, 28(20):E87, Oct 2000.

[4] Ahmadi, A., Behm, A., Honnalli, N., Li, C., Weng, L., et al. Hobbes: optimized gram-based methods for efficient read alignment. *Nucleic Acids Research*, 40(6):e41–e41, 2012.

[5] Albers, C. A., Lunter, G., MacArthur, D. G., McVean, G., Ouwehand, W. H., et al. Dindel: accurate indel calls from short-read data. *Genome Res*, 21(6):961–973, Jun 2011. doi:10.1101/gr.112326.110.

[6] Alkan, C., Kidd, J. M., Marques-Bonet, T., Aksay, G., Antonacci, F., et al. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature genetics*, 41(10):1061–1067, 2009.

[7] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, Oct 1990. doi:10.1016/S0022-2836(05)80360-2.

[8] Ameur, A., Wetterbom, A., Feuk, L., and Gyllensten, U. Global and unbiased detection of splice junctions from RNA-seq data. *Genome Biol*, 11(3):R34, 2010.

[9] Au, K. F., Jiang, H., Lin, L., Xing, Y., and Wong, W. H. Detection of splice junctions from paired-end RNA-seq data by SpliceMap. *Nucleic Acids Research*, 38(14):4570–4578, 2010.

[10] Bao, H., Xiong, Y., Guo, H., Zhou, R., Lu, X., et al. MapNext: a software tool for spliced and unspliced alignments and SNP detection of short sequence reads. *BMC Genomics*, 10 Suppl 3:S13, 2009. doi:10.1186/1471-2164-10-S3-S13.

[11] Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, Nov 2008. doi:10.1038/nature07517.

[12] Berka, J., Chen, Y.-j., Leamon, J. H., Lefkowitz, S., Lohman, K. L., et al. Bead Emulsion Nucleic Acid Amplification, March 28 2013. US Patent 20,130,078,638.

[13] Black, D. L. Mechanisms of alternative pre-messenger RNA splicing. *Annu Rev Biochem*, 72:291–336, 2003. doi:10.1146/annurev.biochem.72.121801.161720.

[14] Bona, F. D., Ossowski, S., Schneeberger, K., and Rtsch, G. Optimal spliced alignments of short sequence reads. *Bioinformatics*, 24(16):i174–i180, Aug 2008. doi:10.1093/bioinformatics/btn300.

[15] Bowers, J., Mitchell, J., Beer, E., Buzby, P. R., Causey, M., et al. Virtual terminator nucleotides for next-generation DNA sequencing. *Nat Methods*, 6(8):593–595, Aug 2009. doi:10.1038/nmeth.1354.

[16] Branton, D., Deamer, D. W., Marziali, A., Bayley, H., Benner, S. A., et al. The potential and challenges of nanopore sequencing. *Nat Biotechnol*, 26(10):1146–1153, Oct 2008. doi:10.1038/nbt.1495.

[17] Brent, M. R. Steady progress and recent breakthroughs in the accuracy of automated genome annotation. *Nat Rev Genet*, 9(1):62–73, Jan 2008. doi:10.1038/nrg2220.

[18] Bryant, D. W., Shen, R., Priest, H. D., Wong, W.-K., and Mockler, T. C. Supersplat–spliced RNA-seq alignment. *Bioinformatics*, 26(12):1500–1505, Jun 2010. doi:10.1093/bioinformatics/btq206.

[19] Burkhardt, S. and Kärkkäinen, J. Better filtering with gapped q-grams. *Fundamenta informaticae*, 56(1):51–70, 2003.

[20] Burrows, M. and Wheeler, D. J. A block-sorting lossless data compression algorithm. 1994.

[21] Chaisson, M. J. and Tesler, G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC bioinformatics*, 13(1):238, 2012.

[22] Chen, Y., Souaiaia, T., and Chen, T. PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. *Bioinformatics*, 25(19):2514–2521, 2009.

[23] Cloonan, N., Xu, Q., Faulkner, G. J., Taylor, D. F., Tang, D. T. P., et al. RNA-MATE: a recursive mapping strategy for high-throughput RNA-sequencing data. *Bioinformatics*, 25(19):2615–2616, Oct 2009. doi:10.1093/bioinformatics/btp459.

[24] Cokus, S. J., Feng, S., Zhang, X., Chen, Z., Merriman, B., et al. Shotgun bisulphite sequencing of the Arabidopsis genome reveals DNA methylation patterning. *Nature*, 452(7184):215–219, Mar 2008. doi:10.1038/nature06745.

[25] Consortium, . G. P., Abecasis, G. R., Altshuler, D., Auton, A., Brooks, L. D., et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, Oct 2010. doi:10.1038/nature09534.

[26] Cox, A. ELAND, 2007.

[27] Denoeud, F., Aury, J.-M., Silva, C. D., Noel, B., Rogier, O., et al. Annotating genomes with massive-scale RNA sequencing. *Genome Biol*, 9(12):R175, 2008. doi:10.1186/gb-2008-9-12-r175.

[28] DiGuistini, S., Liao, N., Platt, D., Robertson, G., Seidel, M., et al. De novo genome sequence assembly of a filamentous fungus using Sanger, 454 and Illumina sequence data. *Genome Biology*, 10(9):R94, 2009. ISSN 1465-6906. doi:10.1186/gb-2009-10-9-r94.

[29] Dimon, M. T., Sorber, K., and DeRisi, J. L. HMMSplicer: a tool for efficient and sensitive discovery of known and novel splice junctions in RNA-Seq data. *PLoS One*, 5(11):e13875, 2010. doi:10.1371/journal.pone.0013875.

[30] Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., et al. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[31] Dressman, D., Yan, H., Traverso, G., Kinzler, K. W., and Vogelstein, B. Transforming single DNA molecules into fluorescent magnetic particles for detection and enumeration of genetic variations. *Proc Natl Acad Sci U S A*, 100(15):8817–8822, Jul 2003. doi:10.1073/pnas.1133470100.

[32] Eckerle, L. D., Becker, M. M., Halpin, R. A., Li, K., Venter, E., et al. Infidelity of SARS-CoV Nsp14-exonuclease mutant virus replication is revealed by complete genome sequencing. *PLoS Pathog*, 6(5):e1000896, May 2010. doi:10.1371/journal.ppat.1000896.

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

[34] Farrar, M. Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23(2):156–161, 2007.

[35] Fedurco, M., Romieu, A., Williams, S., Lawrence, I., and Turcatti, G. BTA, a novel reagent for DNA attachment on glass and efficient generation of solid-phase amplified DNA colonies. *Nucleic acids research*, 34(3):e22–e22, 2006.

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

[37] Fleischmann, R. D., Adams, M. D., White, O., Clayton, R. A., Kirkness, E. F., et al. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science*, 269(5223):496–512, Jul 1995.

[38] Fonseca, N. A., Rung, J., Brazma, A., and Marioni, J. C. Tools for mapping high-throughput sequencing data. *Bioinformatics*, 28(24):3169–3177, 2012.

[39] Garber, M., Grabherr, M. G., Guttman, M., and Trapnell, C. Computational methods for transcriptome annotation and quantification using RNA-seq. *Nat Methods*, 8(6):469–477, Jun 2011. doi:10.1038/nmeth.1613.

[40] Gontarz, P. M., Berger, J., and Wong, C. F. SRmapper: a fast and sensitive genome-hashing alignment tool. *Bioinformatics*, 29(3):316–321, 2013.

[41] Grant, G. R., Farkas, M. H., Pizarro, A. D., Lahens, N. F., Schug, J., et al. Comparative analysis of RNA-Seq alignment algorithms and the RNA-Seq unified mapper (RUM). *Bioinformatics*, 27(18):2518–2528, Sep 2011. doi: 10.1093/bioinformatics/btr427.

[42] Grossi, R. and Vitter, J. S. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

[43] Gudlaugsdottir, S., Boswell, D. R., Wood, G. R., and Ma, J. Exon size distribution and the origin of introns. *Genetica*, 131(3):299–306, Nov 2007. doi:10.1007/s10709-007-9139-4.

[44] Hoffmann, S., Otto, C., Kurtz, S., Sharma, C. M., Khaitovich, P., et al. Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS computational biology*, 5(9):e1000502, 2009.

[45] Homer, N., Merriman, B., and Nelson, S. F. BFAST: an alignment tool for large scale genome resequencing. *PLoS One*, 4(11):e7767, 2009.

[46] Hon, W.-K., Sadakane, K., and Sung, W.-K. Breaking a time-and-space barrier in constructing full-text indices. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 251–260. IEEE, 2003.

[47] Hong, X., Scofield, D. G., and Lynch, M. Intron size, abundance, and distribution within untranslated regions of genes. *Mol Biol Evol*, 23(12):2392–2404, Dec 2006. doi:10.1093/molbev/msl111.

[48] Horner, D. S., Pavesi, G., Castrignan, T., Meo, P. D. D., Liuni, S., et al. Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing. *Brief Bioinform*, 11(2):181–197, Mar 2010. doi:10.1093/bib/bbp046.

[49] Huang, S., Li, R., Zhang, Z., Li, L., Gu, X., et al. The genome of the cucumber, Cucumis sativus L. *Nat Genet*, 41(12):1275–1281, Dec 2009. doi:10.1038/ng.475.

[50] Huang, S., Zhang, J., Li, R., Zhang, W., He, Z., et al. SOAPsplice: Genome-Wide ab initio Detection of Splice Junctions from RNA-Seq Data. *Front Genet*, 2:46, 2011. doi:10.3389/fgene.2011.00046.

[51] Huang, W., Li, L., Myers, J. R., and Marth, G. T. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

[52] Jean, G., Kahles, A., Sreedharan, V. T., Bona, F. D., and Rtsch, G. RNA-Seq read alignments with PALMapper. *Curr Protoc Bioinformatics*, Chapter 11:Unit 11.6, Dec 2010. doi:10.1002/0471250953.bi1106s32.

[53] Jiang, H. and Wong, W. H. SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, 24(20):2395–2396, Oct 2008. doi: 10.1093/bioinformatics/btn429.

[54] Johnson, D. S., Mortazavi, A., Myers, R. M., and Wold, B. Genome-wide mapping of in vivo protein-DNA interactions. *Science*, 316(5830):1497–1502, Jun 2007. doi:10.1126/science.1141319.

[55] Jokinen, P. and Ukkonen, E. Two algorithms for approxmate string matching in static texts. In *Mathematical Foundations of Computer Science 1991*, pages 240–248. Springer, 1991.

[56] Jones, N. C. and Pevzner, P. *An introduction to bioinformatics algorithms*. the MIT Press, 2004.

[57] Karp, R. M. and Rabin, M. O. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[58] Kent, W. J. BLATthe BLAST-like alignment tool. *Genome research*, 12(4):656–664, 2002.

[59] Kim, D., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., et al. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biol*, 14(4):R36, Apr 2013. doi:10.1186/gb-2013-14-4-r36.

[60] Klus, P., Lam, S., Lyberg, D., Cheung, M. S., Pullan, G., et al. BarraCUDA-a fast short read sequence aligner using graphics processing units. *BMC research notes*, 5(1):27, 2012.

[61] Knuth, D. E., Morris, J. H., Jr, and Pratt, V. R. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[62] Ku, C.-S. and Roukos, D. H. From next-generation sequencing to nanopore sequencing technology: paving the way to personalized genomic medicine. *Expert Rev Med Devices*, 10(1):1–6, Jan 2013. doi:10.1586/erd.12.63.

[63] Kurtz, S. The Vmatch large scale sequence analysis software, 2003, 2003.

[64] Kurtz, S., Phillippy, A., Delcher, A. L., Smoot, M., Shumway, M., et al. Versatile and open software for comparing large genomes. *Genome Biol*, 5(2):R12, 2004. doi:10.1186/gb-2004-5-2-r12.

[65] Lam, T. W., Li, R., Tam, A., Wong, S., Wu, E., et al. High throughput short read alignment via bi-directional BWT. In *Bioinformatics and Biomedicine, 2009. BIBM'09. IEEE International Conference on*, pages 31–36. IEEE, 2009.

[66] Lam, T. W., Sung, W. K., Tam, S. L., Wong, C. K., and Yiu, S. M. Compressed indexing and local alignment of DNA. *Bioinformatics*, 24(6):791–797, Mar 2008. doi:10.1093/bioinformatics/btn032.

[67] Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.

[68] Langmead, B. and Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[69] Langmead, B., Trapnell, C., Pop, M., Salzberg, S. L., et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[70] Lee, B. T. K., Tan, T. W., and Ranganathan, S. MGAlignIt: A web service for the alignment of mRNA/EST and genomic sequences. *Nucleic Acids Res*, 31(13):3533–3536, Jul 2003.

[71] Levene, M. J., Korlach, J., Turner, S. W., Foquet, M., Craighead, H. G., et al. Zero-mode waveguides for single-molecule analysis at high concentrations. *Science*, 299(5607):682–686, Jan 2003. doi:10.1126/science.1079700.

[72] Levitt, M. The birth of computational structural biology. *Nat Struct Biol*, 8(5):392–393, May 2001. doi:10.1038/87545.

[73] Li, G., Fullwood, M. J., Xu, H., Mulawadi, F. H., Velkov, S., et al. ChIA-PET tool for comprehensive chromatin interaction analysis with paired-end tag sequencing. *Genome Biol*, 11(2):R22, 2010. doi:10.1186/gb-2010-11-2-r22.

[74] Li, H. and Durbin, R. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[75] Li, H. and Durbin, R. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[76] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, Aug 2009.

[77] Li, H., Ruan, J., and Durbin, R. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–1858, Nov 2008. doi:10.1101/gr.078212.108.

[78] Li, R., Fan, W., Tian, G., Zhu, H., He, L., et al. The sequence and de novo assembly of the giant panda genome. *Nature*, 463(7279):311–317, Jan 2010. doi:10.1038/nature08696.

[79] Li, R., Li, Y., Kristiansen, K., and Wang, J. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

[80] Li, R., Yu, C., Li, Y., Lam, T.-W., Yiu, S.-M., et al. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[81] Li, Y., Li-Byarlay, H., Burns, P., Borodovsky, M., Robinson, G. E., et al. TrueSight: a new algorithm for splice junction detection using RNA-seq. *Nucleic Acids Res*, 41(4):e51, Feb 2013. doi:10.1093/nar/gks1311.

[82] Lim, J.-Q., Tennakoon, C., Li, G., Wong, E., Ruan, Y., et al. BatMeth: improved mapper for bisulfite sequencing reads on DNA methylation. *Genome Biol*, 13(10):R82, Oct 2012. doi:10.1186/gb-2012-13-10-r82.

[83] Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., and Li, M. ZOOM! Zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437, Nov 2008. doi:10.1093/bioinformatics/btn416.

[84] Liu, C.-M., Wong, T., Wu, E., Luo, R., Yiu, S.-M., et al. SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics*, 28(6):878–879, 2012.

[85] Liu, L., Li, Y., Li, S., Hu, N., He, Y., et al. Comparison of next-generation sequencing systems. *J Biomed Biotechnol*, 2012:251364, 2012. doi:10.1155/2012/251364.

[86] Liu, Y. and Schmidt, B. Long read alignment based on maximal exact match seeds. *Bioinformatics*, 28(18):i318–i324, 2012.

[87] Liu, Y., Schmidt, B., and Maskell, D. L. CUDASW++ 2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC research notes*, 3(1):93, 2010.

[88] Liu, Y., Schmidt, B., and Maskell, D. L. CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows–Wheeler transform. *Bioinformatics*, 28(14):1830–1837, 2012.

[89] Lunter, G. and Goodson, M. Stampy: a statistical algorithm for sensitive and fast mapping of Illumina sequence reads. *Genome research*, 21(6):936–939, 2011.

[90] Luo, R., Wong, T., Zhu, J., Liu, C.-M., Zhu, X., et al. SOAP3-dp: Fast, Accurate and Sensitive GPU-Based Short Read Aligner. *PloS one*, 8(5):e65632, 2013.

[91] Ma, B., Tromp, J., and Li, M. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[92] Magi, A., Benelli, M., Gozzini, A., Girolami, F., Torricelli, F., et al. Bioinformatics for next generation sequencing data. *Genes*, 1(2):294–307, 2010.

[93] Manber, U. and Myers, G. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[94] Marco-Sola, S., Sammeth, M., Guigó, R., and Ribeca, P. The GEM mapper: fast, accurate and versatile alignment by filtration. *Nature methods*, 9(12):1185–1188, 2012.

[95] Mardis, E. R. Next-Generation Sequencing Platforms. *Annu Rev Anal Chem (Palo Alto Calif)*, Apr 2013. doi:10.1146/annurev-anchem-062012-092628.

[96] Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, Sep 2005. doi:10.1038/nature03959.

[97] Markljung, E., Jiang, L., Jaffe, J. D., Mikkelsen, T. S., Wallerman, O., et al. ZBED6, a novel transcription factor derived from a domesticated DNA transposon regulates IGF2 expression and muscle growth. *PLoS Biol*, 7(12):e1000256, Dec 2009. doi:10.1371/journal.pbio.1000256.

[98] Maxam, A. M. and Gilbert, W. A new method for sequencing DNA. *Proc Natl Acad Sci U S A*, 74(2):560–564, Feb 1977.

[99] McCreight, E. M. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2):262–272, 1976.

[100] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*, 20(9):1297–1303, Sep 2010. doi:10.1101/gr.107524.110.

[101] Mckernan, K., Blanchard, A., Kotler, L., and Costa, G. Reagents, methods, and libraries for bead-based sequencing, December 2 2009. US Patent App. 12/629,858.

[102] McKernan, K. J., Peckham, H. E., Costa, G. L., McLaughlin, S. F., Fu, Y., et al. Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. *Genome Res*, 19(9):1527–1541, Sep 2009. doi:10.1101/gr.091868.109.

[103] Meek, C., Patel, J. M., and Kasetty, S. Oasis: An online and accurate technique for local-alignment searches on biological sequences. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 910–921. VLDB Endowment, 2003.

[104] Mikkelsen, T. S., Ku, M., Jaffe, D. B., Issac, B., Lieberman, E., et al. Genome-wide maps of chromatin state in pluripotent and lineage-committed cells. *Nature*, 448(7153):553–560, Aug 2007. doi:10.1038/nature06008.

[105] Mikkelsen, T. S., Wakefield, M. J., Aken, B., Amemiya, C. T., Chang, J. L., et al. Genome of the marsupial Monodelphis domestica reveals innovation in non-coding sequences. *Nature*, 447(7141):167–177, May 2007. doi:10.1038/nature05805.

[106] Mills, R. E., Luttig, C. T., Larkins, C. E., Beauchamp, A., Tsui, C., et al. An initial map of insertion and deletion (INDEL) variation in the human genome. *Genome Res*, 16(9):1182–1190, Sep 2006. doi:10.1101/gr.4565806.

[107] Mitra, R. D., Shendure, J., Olejnik, J., Edyta-Krzymanska-Olejnik, and Church, G. M. Fluorescent in situ sequencing on polymerase colonies. *Anal Biochem*, 320(1):55–65, Sep 2003.

[108] Morin, R., Bainbridge, M., Fejes, A., Hirst, M., Krzywinski, M., et al. Profiling the HeLa S3 transcriptome using randomly primed cDNA and massively parallel short-read sequencing. *BioTechniques*, 45(1):81–94, July 2008. ISSN 0736-6205. doi:10.2144/000112900.

[109] Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., and Wold, B. Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nat Methods*, 5(7):621–628, Jul 2008. doi:10.1038/nmeth.1226.

[110] Mu, J. C., Jiang, H., Kiani, A., Mohiyuddin, M., Asadi, N. B., et al. Fast and accurate read alignment for resequencing. *Bioinformatics*, 28(18):2366–2373, 2012.

[111] Myers, E. W. AnO (ND) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

[112] Nagalakshmi, U., Wang, Z., Waern, K., Shou, C., Raha, D., et al. The transcriptional landscape of the yeast genome defined by RNA sequencing. *Science*, 320(5881):1344–1349, 2008. doi:10.1126/science.1158441.

[113] Needleman, S. B. and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, Mar 1970.

[114] Ng, S. B., Buckingham, K. J., Lee, C., Bigham, A. W., Tabor, H. K., et al. Exome sequencing identifies the cause of a mendelian disorder. *Nature Genetics*, 42(1):30–35, November 2009. ISSN 1061-4036. doi:10.1038/ng.499.

[115] Ning, Z., Cox, A. J., and Mullikin, J. C. SSAHA: a fast search method for large DNA databases. *Genome Res*, 11(10):1725–1729, Oct 2001. doi:10.1101/gr.194201.

[116] Pan, Q., Shai, O., Lee, L. J., Frey, B. J., and Blencowe, B. J. Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing. *Nat Genet*, 40(12):1413–1415, Dec 2008. doi:10.1038/ng.259.

[117] Quail, M. A., Smith, M., Coupland, P., Otto, T. D., Harris, S. R., et al. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics*, 13:341, 2012. doi:10.1186/1471-2164-13-341.

[118] Rognes, T. and Seeberg, E. Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.

[119] Ronaghi, M., Uhln, M., and Nyrn, P. A sequencing method based on real-time pyrophosphate. *Science*, 281(5375):363, 365, Jul 1998.

[120] Rothberg, J. M., Hinz, W., Rearick, T. M., Schultz, J., Mileski, W., et al. An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, 475(7356):348–352, Jul 2011. doi:10.1038/nature10242.

[121] Ruffalo, M., Koyutrk, M., Ray, S., and LaFramboise, T. Accurate estimation of short read mapping quality for next-generation genome sequencing. *Bioinformatics*, 28(18):i349–i355, Sep 2012. doi:10.1093/bioinformatics/bts408.

[122] Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., et al. SHRiMP: accurate mapping of short color-space reads. *PLoS computational biology*, 5(5):e1000386, 2009.

[123] Ryan, M. C., Cleland, J., Kim, R., Wong, W. C., and Weinstein, J. N. SpliceSeq: a resource for analysis and visualization of RNA-Seq data on alternative splicing and its functional impacts. *Bioinformatics*, 28(18):2385–2387, 2012.

[124] Sakharkar, M. K., Chow, V. T. K., and Kangueane, P. Distributions of exons and introns in the human genome. *In Silico Biol*, 4(4):387–393, 2004.

[125] Sanger, F., Air, G. M., Barrell, B. G., Brown, N. L., Coulson, A. R., et al. Nucleotide sequence of bacteriophage phi X174 DNA. *Nature*, 265(5596):687–695, Feb 1977.

[126] Sanger, F., Nicklen, S., and Coulson, A. R. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A*, 74(12):5463–5467, Dec 1977.

[127] Schmucker, D., Clemens, J. C., Shu, H., Worby, C. A., Xiao, J., et al. Drosophila Dscam is an axon guidance receptor exhibiting extraordinary molecular diversity. *Cell*, 101(6):671–684, Jun 2000.

[128] Schneeberger, K., Hagmann, J., Ossowski, S., Warthmann, N., Gesing, S., et al. Simultaneous alignment of short reads against multiple genomes. *Genome Biol*, 10(9):R98, 2009. doi:10.1186/gb-2009-10-9-r98.

[129] Shendure, J., Porreca, G. J., Reppas, N. B., Lin, X., McCutcheon, J. P., et al. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, 309(5741):1728–1732, Sep 2005. doi:10.1126/science.1117389.

[130] Siragusa, E., Weese, D., and Reinert, K. Fast and accurate read mapping with approximate seeds and multiple backtracking. *Nucleic acids research*, 41(7):e78–e78, 2013.

[131] Slater, G. S. C. and Birney, E. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, 6:31, 2005. doi:10.1186/1471-2105-6-31.

[132] Smith, A. D., Xuan, Z., and Zhang, M. Q. Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC bioinformatics*, 9(1):128, 2008.

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

[134] Sung, W.-K., Zheng, H., Li, S., Chen, R., Liu, X., et al. Genome-wide survey of recurrent HBV integration in hepatocellular carcinoma. *Nat Genet*, 44(7):765–769, Jul 2012. doi:10.1038/ng.2295.

[135] Tennakoon, C., Purbojati, R. W., and Sung, W.-K. BatMis: a fast algorithm for k-mismatch mapping. *Bioinformatics*, 28(16):2122–2128, Aug 2012. doi:10.1093/bioinformatics/bts339.

[136] Thompson, J. F. and Steinmann, K. E. Single molecule sequencing with a HeliScope genetic analysis system. *Curr Protoc Mol Biol*, Chapter 7:Unit7.10, Oct 2010. doi:10.1002/0471142727.mb0710s92.

[137] Trapnell, C., Pachter, L., and Salzberg, S. L. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, May 2009. doi:10.1093/bioinformatics/btp120.

[138] Trapnell, C., Williams, B. A., Pertea, G., Mortazavi, A., Kwan, G., et al. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat Biotechnol*, 28(5):511–515, May 2010. doi:10.1038/nbt.1621.

[139] Travers, K. J., Chin, C.-S., Rank, D. R., Eid, J. S., and Turner, S. W. A flexible and efficient template format for circular consensus sequencing and SNP detection. *Nucleic Acids Res*, 38(15):e159, Aug 2010. doi:10.1093/nar/gkq543.

[140] Treffer, R. and Deckert, V. Recent advances in single-molecule sequencing. *Current Opinion in Biotechnology*, 21(1):4 – 11, 2010. ISSN 0958-1669. doi:http://dx.doi.org/10.1016/j.copbio.2010.02.009. ¡ce:title¿Analytical Biotechnology¡/ce:title¿.

[141] Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, Feb 2001. doi:10.1126/science.1058040.

[142] Wang, K., Singh, D., Zeng, Z., Coleman, S. J., Huang, Y., et al. MapSplice: accurate mapping of RNA-seq reads for splice junction discovery. *Nucleic acids research*, 38(18):e178–e178, 2010.

[143] Wang, L., Wang, X., Wang, X., Liang, Y., and Zhang, X. Observations on novel splice junctions from RNA sequencing data. *Biochem Biophys Res Commun*, 409(2):299–303, Jun 2011. doi:10.1016/j.bbrc.2011.05.005.

[144] Wang, Z., Gerstein, M., and Snyder, M. RNA-Seq: a revolutionary tool for transcriptomics. *Nat Rev Genet*, 10(1):57–63, Jan 2009. doi:10.1038/nrg2484.

[145] Weese, D., Emde, A.-K., Rausch, T., Döring, A., and Reinert, K. RazerSfast read mapping with sensitivity control. *Genome research*, 19(9):1646–1654, 2009.

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

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

[148] Wheeler, D. A., Srinivasan, M., Egholm, M., Shen, Y., Chen, L., et al. The complete genome of an individual by massively parallel DNA sequencing. *Nature*, 452(7189):872–876, Apr 2008. doi:10.1038/nature06884.

[149] Wolf, A. B., Caselli, R. J., Reiman, E. M., and Valla, J. APOE and neuroenergetics: an emerging paradigm in Alzheimer's disease. *Neurobiol Aging*, 34(4):1007–1017, Apr 2013. doi:10.1016/j.neurobiolaging.2012.10.011.

[150] Wood, D. L. A., Xu, Q., Pearson, J. V., Cloonan, N., and Grimmond, S. M. X-MATE: a flexible system for mapping short read data. *Bioinformatics*, 27(4):580–581, Feb 2011. doi:10.1093/bioinformatics/btq698.

[151] Wu, J., Anczukw, O., Krainer, A. R., Zhang, M. Q., and Zhang, C. OLego: fast and sensitive mapping of spliced mRNA-Seq reads using small seeds. *Nucleic Acids Res*, 41(10):5149–5163, May 2013. doi:10.1093/nar/gkt216.

[152] Wu, T. D. and Nacu, S. Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*, 26(7):873–881, Apr 2010. doi:10.1093/bioinformatics/btq057.

[153] Wu, T. D. and Watanabe, C. K. GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics*, 21(9):1859–1875, May 2005. doi:10.1093/bioinformatics/bti310.

[154] Zhang, J., Chiodini, R., Badr, A., and Zhang, G. The impact of next-generation sequencing on genomics. *J Genet Genomics*, 38(3):95–109, Mar 2011. doi:10.1016/j.jgg.2011.02.003.

[155] Zhang, Y., Lameijer, E.-W., 't Hoen, P. A. C., Ning, Z., Slagboom, P. E., et al. PASSion: a pattern growth algorithm-based pipeline for splice junction detection in paired-end RNA-Seq data. *Bioinformatics*, 28(4):479–486, Feb 2012. doi:10.1093/bioinformatics/btr712.

[156] Zhang, Z., Schwartz, S., Wagner, L., and Miller, W. A greedy algorithm for aligning DNA sequences. *Journal of Computational biology*, 7(1-2):203–214, 2000.

[157] Zhang, Z. D., Du, J., Lam, H., Abyzov, A., Urban, A. E., et al. Identification of genomic indels and structural variations using split reads. *BMC genomics*, 12(1):375, 2011.

[158] Zhao, M., Lee, W.-P., and Marth, G. T. SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. *arXiv preprint arXiv:1208.6350*, 2012.

Appendix A

Additional Mapping Results

A.1 List of Publications

A.1.1 Journal Publications

1. BatMis: A fast algorithm for k-mismatch mapping (Bioinformatics, 2012)- Chandana Tennakoon, Rikky W. Purbojati and Wing-Kin Sung

2. BatMeth: improved mapper for bisulfite sequencing reads on DNA methylation.(Genome Biology, 2012)- Jing-Quan Lim, Chandana Tennakoon, Guoliang Li, Eleanor Wong, Yijun Ruan, Chia-Lin Wei and Wing-Kin Sung

3. Genome-wide survey of recurrent HBV integration in hepatocellular carcinoma (Nature Genetics, 2012)- Wing-Kin Sung, Hancheng Zheng, Shuyu Li, Ronghua Chen, Xiao Liu, Yingrui Li, Nikki P Lee, Wah H Lee, Pramila N Ariyaratne, Chandana Tennakoon, Fabianus H Mulawadi, Kwong F Wong, Angela M Liu, Ronnie T Poon, Sheung Tat Fan, Kwong L Chan, Zhuolin Gong, Yujie Hu, Zhao Lin, Guan Wang, Qinghui Zhang, Thomas D Barber, Wen-Chi Chou, Amit Aggarwal, Ke Hao, Wei Zhou, Chunsheng Zhang, James Hardwick, Carolyn Buser, Jiangchun Xu, Zhengyan Kan, Hongyue Dai, Mao Mao, Christoph Reinhard, Jun

Wang and John M Luk

4. ChIA-PET tool for comprehensive chromatin interaction analysis with paired-end tag sequencing (Genome Biology, 2010)- Guoliang Li,, Melissa J Fullwood, Han Xu, Fabianus Hendriyan Mulawadi, Stoyan Velkov, Vinsensius Vega, Pramila Nuwantha Ariyaratne, Yusoff Bin Mohamed, Hong-Sain Ooi, Chandana Tennakoon, Chia-Lin Wei, Yijun Ruan and Wing-Kin Sung

A.1.2 Poster Presentations

1. Fast and Accurate Alignment with BatAlign (International Conference on Genome Informatics 2013) - Jing-Quan Lim, Chandana Tennakoon and Wing-Kin Sung

A.2 Additional Mapping Results

We have given below some additional mapping results for BatMis.

A.3 Software

The source code for the software can be downloaded from the following locations:

1. BatMis: https://code.google.com/p/batmis/
2. BatAlign: https://bitbucket.org/drcyber/batindel/
3. BatRNA: https://bitbucket.org/drcyber/rnaseq/

Test data sets for BatAlign and BatRNA can be found in http://compbio.ddns.comp.nus/ limjingq/RNA and http://compbio.ddns.comp.nus/ limjingq/BATALIGN.

	1-mis	2mis	3-mis	4-mis	5-mis
BatMis	0	0	0	0	0
BWA	26	48	94	124	168
ZOOM	30	72	107	147	
RazerS2	9	9	9	9	9

Table A.1: *Number of incorrect multiple mappings reported by aligners for different numbers of mismatches. BatMis does not report any incorrect hits.*

		2-mis	3-mis	4-mis	5-mis
51bp	BWA	0	467	1400	2540
	RazerS2	119	107	185	93
100bp	BWA	48	167	1592	2782
	Razers2	0	0	4	22

Table A.2: *Number of incorrect unique hits reported by BWA and Razers2 for different numbers of mismatches when run in their heuristic modes.*

		0 Mis	1 Mis	2 Mis	3 Mis	4 Mis	5 Mis
51bp	Batmis	100000	100000	100000	100000	100000	100000
	BWA	100000	100000	100000	100000	100000	100000
	ZOOM	100000	100000	100000	100000	100000	
	Razers2	100000	100000	100000	100000	100000	100000

		0 Mis	1 Mis	2 Mis	3 Mis	4 Mis	5 Mis	8 Mis	10 Mis
100bp	Batmis	100000	100000	100000	100000	100000	100000	100000	100000
	BWA	100000	100000	100000	100000	100000	100000	97291	15124
	ZOOM	100000	100000	100000	100000	100000			
	Razers2	100000	100000	100000	100000	100000	100000	100000	100000

Table A.3: *Number of least mismatch hits reported by aligners when mapping simulated k-mismatch datasets containing 100 000 reads. Ideally, each program should report 100 000 hits.*

		2 Mis	**3 Mis**	**4 Mis**	**5 Mis**
100bp	BatAlign	31001496	59121695	96502481	143315831
	BWA	31001496	58148475	91804113	130088604
		1 Mis	**2 Mis**	**3 Mis**	**4 Mis**
51bp	BatAlign	87004377	221353752	482270385	972545515
	BWA	87004377	221353752	426489802	688550754

Table A.4: *Number of multiple mappings reported by BWA in its heuristic mode and with the exact algorithm of BatMis for a 100bp dataset containing 1 000 000 reads.*