Bioinformatics Challenges of High-Throughput SNP Discovery and Utilization in non-Model Organisms

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Wayne E. Clarke

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

A current trend in biological science is the increased use of computational tools for both the production and analysis of experimental data. This is especially true in the field of genomics, where advancements in DNA sequencing technology have dramatically decreased the time and cost associated with DNA sequencing resulting in increased pressure on the time required to prepare and analyze data generated during these experiments. As a result, the role of computational science in such biological research is increasing.

This thesis seeks to address several major questions with respect to the development and application of single nucleotide polymorphism (SNP) resources in non-model organisms. Traditional SNP discovery using polymerase chain reaction (PCR) amplification and low-throughput DNA sequencing is a time consuming and laborious process, which is often limited by the time required to design intron-spanning PCR primers. While next-generation DNA sequencing (NGS) has largely supplanted low-throughput sequencing for SNP discovery applications, the PCR based SNP discovery method remains in use for cost effective, targeted SNP discovery. This thesis seeks to develop an automated method for intron-spanning PCR design which would remove a significant bottleneck in this process. This work develops algorithms for combining SNP data from multiple individuals, independent of the DNA sequencing platforms, for the purpose of developing SNP genotyping arrays. Additionally, tools for the filtering and selection of SNPs will be developed, providing start to finish support for the development of SNP genotyping arrays in complex polyploids using NGS.

The result of this work includes two automated pipelines for the design of intron-spanning PCR primers, one which designs a single primer pair per target and another that designs multiple primer pairs per target. These automated pipelines are shown to reduce the time required to design primers from one hour per primer pair using the semi-automated method to 10 minutes per 100 primer pairs while maintaining a very high efficacy. Efficacy is tested by comparing the number of successful PCR amplifications of the semi-automated method with that of the automated pipelines. Using the Chi-squared test, the semi-automated and automated approaches are determined not to differ in efficacy.

Three algorithms for combining SNP output from NGS data from multiple individuals are developed and evaluated for their time and space complexities. These algorithms were found to be computationally efficient, requiring time and space linear to the size of the input. These algorithms are then implemented in the Perl language and their time and memory performance profiled using experimental data. Profiling results are evaluated by applying linear models, which allow for predictions of resource requirements for various input sizes. Additional tools for the filtering of SNPs and selection of SNPs for a SNP array are developed and applied to the creation of two SNP arrays in the polyploid crop *Brassica napus*. These arrays, when compared to arrays in similar species, show higher numbers of polymorphic markers and better 3-cluster genotype separation, a viable method for determining the efficacy of design in complex genomes.

Acknowledgements

First and foremost I would like to thank my supervisors Drs. Ian McQuillan and Isobel Parkin for the opportunity to study under them. Their commitment to excellence in their research inspires me. I would also like to thank the members of my advisory committee–Drs. Tony Kusalik, Michael Horsch, and Steve Robinson–for taking the time to serve on my committee and Dr. Josh Udall for serving as my external examiner. Their advice and contributions are greatly appreciated. I would also like to recognize the contributions and support of Dr. Andrew Sharpe, Christine Sidebottom, and Erin Higgins.

Funding for the projects presented in this thesis were provided by the Brassica SNP Consortium and the CanSeq Brassica Sequencing Initiative.

Finally, I would like to thank my family and friends for all of their kind words of support.

To my wife,

For all of the love you have shown me,

for all of your encouragement and support,

I lovingly dedicate this thesis to you.

I couldn't have done it without you.

Contents

List of Tables

List of Figures

List of Abbreviations

APS	Adenylyl Sulfate
ATP	Adenosine Tri-Phosphate
BAC	Bacterial Artificial Chromosome
BAM	Binary Alignment/Map
BLAST	Basic Local Alignment Search Tool
BLAT	BLAST-Like Alignment Tool
CCD	Charge-Coupled Device
cDNA	Complementary DNA
CIGAR	Compact Idiosyncratic Gapped Alignment Report
DNA	Deoxyribonucleic Acid
EST	Expressed Sequence Tag
HSP	High Scoring Pairs
IUPAC	International Union of Pure and Applied Chemistry
mRNA	Messenger RNA
NCBI	National Center for Biotechnology Information
NGS	Next-generation Sequencing
PCR	Polymerase Chain Reaction
PP_i	Pyrophosphate
RAD	Restriction Site Associated DNA
RNA	Ribonucleic Acid
rtPCR	Real-Time PCR
SAM	Sequence Alignment/Map
SBS	Sequencing By Synthesis
SNP	Single Nucleotide Polymorphism
SRA	Sequence Read Archive
SSR	Simple Sequence Repeat
UTR	Untranslated Region
VCF	Variant Call Format

Chapter 1

Introduction

Due to recent technological advancements, researchers in multiple fields of biological science are designing far more complex and expansive experiments, which is rapidly increasing the role of computational science in supporting the generation and analysis of the research. This is especially true in the field of genomics, where DNA sequencing technologies are advancing rapidly. In fact, advances in DNA sequencing technologies are currently outpacing Moore's Law [124]. Moore's Law is an observation that "compute power" doubles every two years. Moore's Law is often used for measuring the progress of technological innovation in many fields, where technologies that keep up with Moore's Law are regarded as performing very well. Figure 1.1 shows the actual cost of sequencing 1 megabase of DNA between September of 2001 and January of 2013 compared to the hypothetical cost of sequencing 1 megabase as predicted by Moore's Law (equating sequencing throughput to compute power and assuming that the doubling of compute power occurs at the same cost).

As a result of decreases in sequencing cost, there has been an exponential increase in the amount of available sequence information as evidenced by the growth of the National Center for Biotechnology Information's (NCBI) Sequence Read Archive (Figure 1.2) [31]. The vast wealth of biological information that can potentially be extracted from this sequence information cannot be discovered without the use of computational methods.

Capturing natural variation between and within species has applications in many areas of biological study for both mammalian and plant systems. The most commonly found genome-wide variations are single nucleotide polymorphisms (SNPs) [27, 24, 18]. SNPs are single base changes between two closely related DNA sequences (Figure 1.3). DNA sequencing is an important technology for the discovery of SNPs, as the resulting DNA sequences can be mined to isolate and characterize these variations. Over the course of this work, advances in DNA sequencing technologies allowed for new techniques with regards to SNP discovery in non-model organisms. This necessitated a change in the direction of my research from SNP discovery using Sanger based sequencing approaches to the use of next generation sequencers. Early adoption of the next generation sequencing technologies, as well as the volume of data they are capable of producing, provided many challenges.

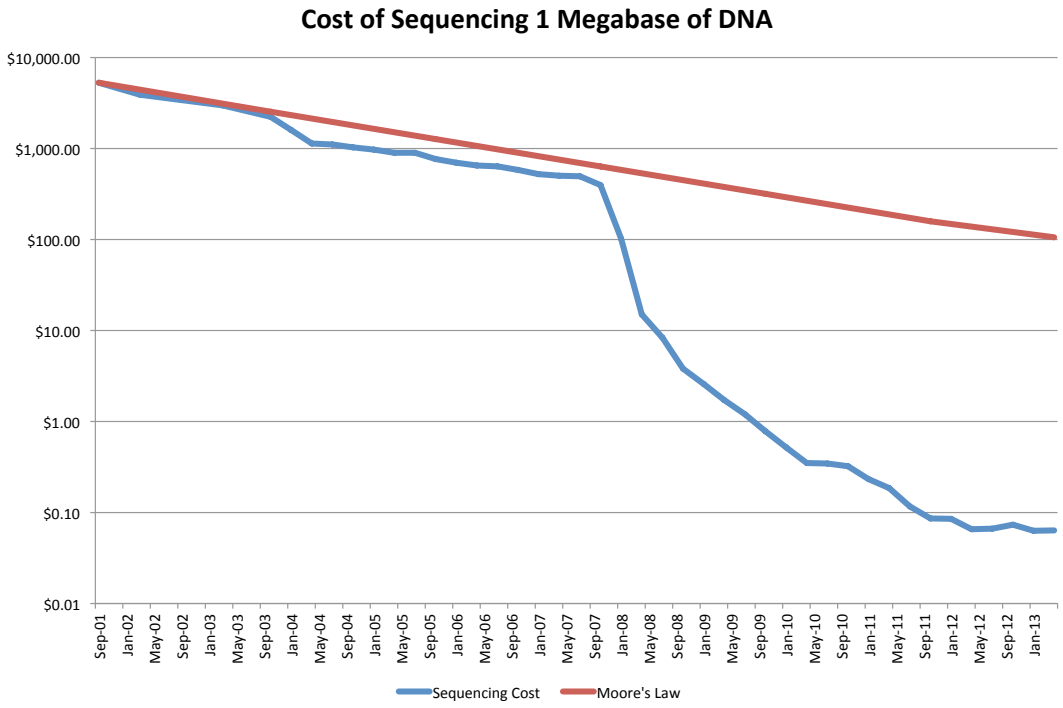

Figure 1.1: Cost of DNA sequencing (y-axis, logarithmic) over time, as measured (blue) and predicted by Moore's Law (red). The rapid decrease in the actual cost of DNA sequencing is a direct correlation to the advancements made in DNA sequencing technologies. Data for DNA sequencing cost obtained from [124].

1.1 Motivation and Objectives

1.1.1 Motivation

SNPs are the most abundant variation found in the DNA between two individuals. Because of their abundance, SNPs are one of the most important tools available to researchers studying genetic differences. Applications of SNPs range from human health, such as the study of cancer or genetic disorders, to plant breeding [24]. Other advantages of utilizing SNPs in genetic research are that the evolutionary processes which create them are well understood and they can be easily and cheaply compared by different laboratories [26].

All of Canada's important commercial crops are considered non-model organisms and therefore have not received as much scientific study as model organisms such as *Arabidopsis thaliana*. This includes crops such as canola, spring and durum wheat, barley, and soybeans. In Canada, canola (*Brassica napus*) alone is a $19.3 billion/year industry, making it an important contributor to the Canadian economy [14]. As such, developing resources to aid plant breeding programs in the improvement of these crops will be key to the continued success of agriculture in Canada and throughout the world.

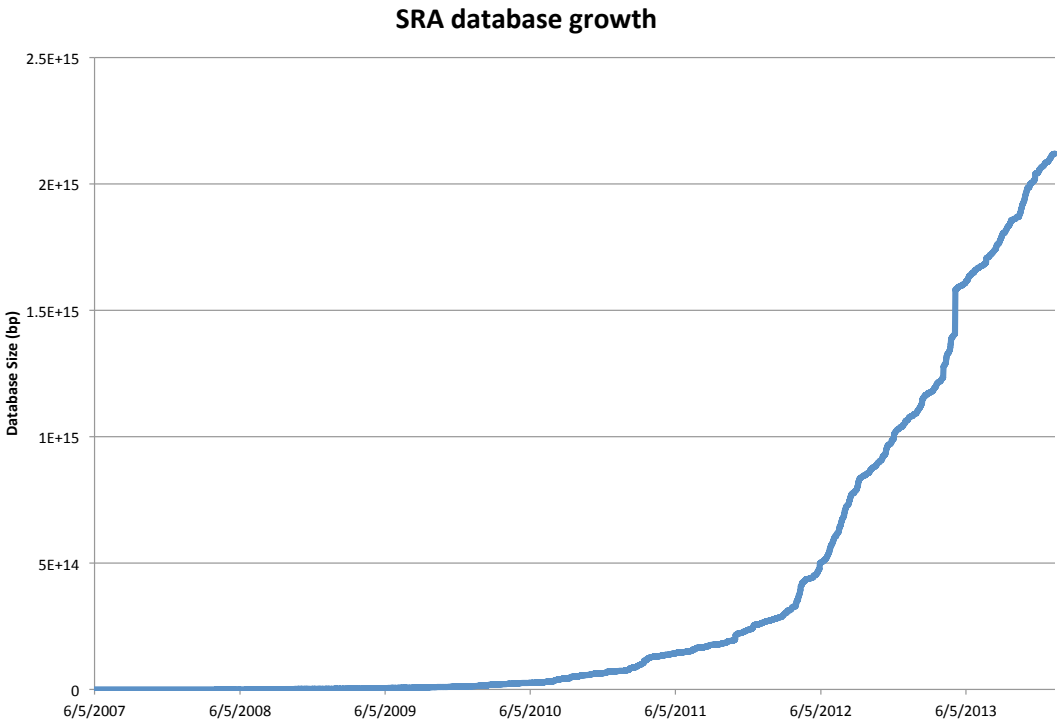

Figure 1.2: Exponential growth of the NCBI Sequence Read Archive (SRA) which was created to hold publicly available sequence data from next generation sequencers. Data obtained from NCBI [31].

1.1.2 Objectives

The goal of this work is to create new computational methods for the development and utilization of SNP resources for non-model organisms. Our primary objectives are to:

1. Automate the design of intron-spanning PCR primers in non-model organisms in order to remove a significant bottleneck to SNP discovery using first generation DNA sequencing. This requires maintaining the efficacy of non-automated approaches while significantly reducing primer design time.

2. Design and implement algorithms that can combine SNP data from multiple individuals and whose output provides sufficient biological information to allow effective design of a robust SNP genotyping array in species with complex genomes.

3. Evaluate the computational complexity and performance of developed algorithms. Computational complexity will be evaluated with respect to time and space (asymptotic time and space complexity, using $\mathcal{O}$-notation, Ω-notation, and Θ-notation). Performance will also be evaluated by varying the size of the input and collecting data on the running time and memory usage. Performance data will

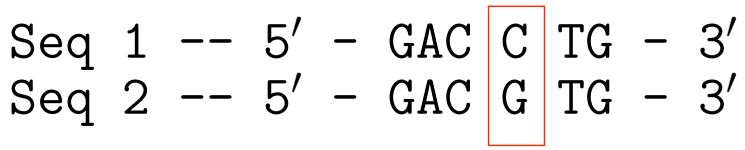

Figure 1.3: An example of a simple SNP at base position 4 between two related individuals.

then be evaluated using a regression model and compared to the theoretical results obtained from the complexity analysis.

4. Design a SNP genotyping array and compare results obtained from it to results from other genotyping arrays in the literature. Array design will include the development of computational tools for filtering the large amount of SNP output and selecting SNPs to be included on the genotyping array.

1.2 Thesis Overview

In this thesis, computational methods will be described for the discovery and utilization of SNPs in non-model organisms. The success of these methods is dependent on the underlying genome structure and organization of the species being investigated as well as the laboratory methods used to generate the data. Chapter 2 introduces important background concepts with respect to both the computer science and the biology pertinent to this thesis. In Chapter 3, a method is described for automating a key bottleneck for SNP discovery using first generation DNA sequencing. The time required by the automated method will be compared to the non-automated method and the resulting output statistically evaluated.

Chapter 4, describes the development of algorithms for combining SNP data from multiple individuals, a key step in the design of SNP genotyping arrays for both diploid and complex polyploid genomes (Sections 4.5 and 4.6). Moreover, these algorithms are computationally evaluated with respect to their theoretical time and space complexity to determine their suitability for processing large genomic data sets (Section 4.7). The time and space requirements of each algorithm is further experimentally evaluated using varying input sizes and performing linear regression analysis (Section 4.9). A discussion of potential bottlenecks in the parallelization is given in Section 4.10 followed by a discussion on the selection of an appropriate algorithm for a given input (Section 4.11). Methods for the filtering of the combined SNP data based on several criteria and selection of SNPs are developed in Sections 4.12 and 4.13, respectively. Finally, results from the application of the developed methods are compared to results available in the literature (Section 4.14) and an overall discussion of the chapter performed (Section 4.15). Chapter 5 will then discuss the results of the thesis as a whole and conclusions that can be drawn from this work. Further, it will discuss the potential directions that this work might take in the future.

Chapter 2

Relevant Background

2.1 Introduction

This chapter will introduce the basic concepts and techniques important to the discussion of the thesis work. Section 2.2 describes the basic biological terminology and techniques used throughout the thesis. As the focus of this thesis is on the development of computational resources for SNP discovery and utilization, Section 2.3 provides some motivation and details regarding several common research applications of SNPs. A brief overview of DNA sequencing (Section 2.4) is provided, followed by an introduction of the relevant bioinformatics and computer science approaches (Section 2.5). Sections 2.6, 2.7, and 2.8 describe common SNP discovery methods, DNA sequencing technologies, and DNA sequencing methodologies respectively.

2.2 Biological Background

An organism's DNA, often referred to as its genome, contains much of the information that makes that organism unique and is the primary method for inheritance. DNA is made up of two complementary strands, which consist of many nucleotides (the building blocks of DNA). There are four nucleotides that make up DNA sequences: adenine, guanine, cytosine, and thymine (A, G, C, and T respectively). The four nucleotides can be subdivided into two groups, purines (adenine and guanine) and pyrimidines (cytosine and thymine). In DNA molecules, the nucleotides adenine and thymine are complementary as are guanine and cytosine. Therefore, if one strand of DNA is a chain of guanine, adenine, cytosine, cytosine, thymine, and guanine (represented by the sequence GACCTG), the complementary strand would be cytosine, thymine, guanine, guanine, adenine, and cytosine (CTGGAC). DNA sequences have terminal ends that are labelled as either $3'$ or $5'$. Since the strands are complementary the result is that the $3'$ end of one strand is matched with the $5'$ end of the other, and for this reason the second strand is referred to as the reverse complement (Figure 2.1) [97].

$$5' - GACCTG - 3'$$
$$3' - CTGGAC - 5'$$

Figure 2.1: The reverse complement (CAGGTC) of the sequence GACCTG, illustrating the pairing of the 5′ and 3′ ends.

2.2.1 Genome Structure and Organization

An organism's DNA is organized into one or more structures called chromosomes. Prokaryotes are organisms without a nucleus, such as bacteria, and generally contain a single chromosome. In contrast, eukaryotes have a number of chromosomes contained in the nucleus, where the number of chromosomes depends on the species [97].

Within the chromosomes are substrings of DNA called genes. In eukaryotes, genes are composed of subsequences known as exons, introns, and untranslated regions (UTRs) (Figure 2.2a.). Genes found in DNA can be converted into ribonucleic acid (RNA), a shorter term storage medium than DNA, during a process called transcription. These RNAs are often referred to as transcripts. Messenger RNA (mRNA) is a RNA molecule that is used to make proteins. During the process of translation, the information contained in the mRNA is read to determined the protein's structure. The introns are ultimately removed during conversion of the gene to messenger RNA (Figure 2.2b.). During translation the UTRs regions are ignored, thus the name untranslated region. Exploiting this biology, complementary DNA (cDNA), which are DNA molecules that are complementary to a mRNA molecule and therefore represent the DNA sequence of a functional protein, can be generated. An expressed sequence tag (EST) is a small subsequence of a cDNA molecule [64].

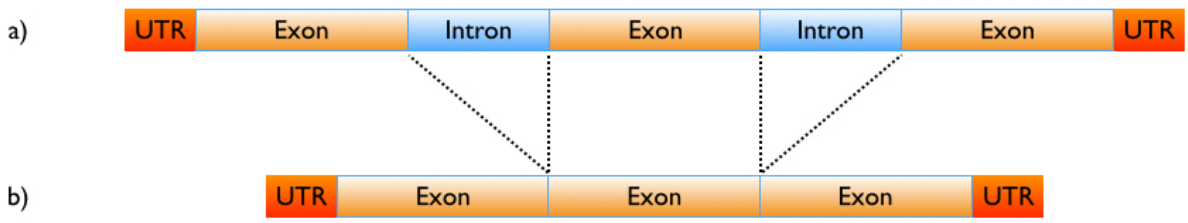

Figure 2.2: A representation of the gene structure (a) including exons, introns, and UTRs and of the converted mRNA (b) with introns removed.

Haploids, Diploids, and Polyploids

Organisms can be classified into one of three categories (haploid, diploid, or polyploid) based on the number of copies of each chromosome found in each somatic cell (a cell which makes up the body of the organism). Diploid species, such as humans, have two copies of each chromosome (2N) while haploid species contain only a single copy of each chromosome (N) [97].

Polyploids are species that have more than two complete sets of chromosomes. For example, an organism with 4 sets of chromosomes are known as tetraploids and those with 6 sets of chromosomes as hexaploids. Although diploids are most common for mammals, polyploidy is a common occurrence in nature, particularly in plants, and results from whole genome duplication events. There are two types of duplication events: those resulting from the fusion of two genomes of the same species (autopolyploids) and those resulting from the fusion of two separate but related species (allopolyploids) [125]. These genome fusion events result in multiple copies of highly related genes, increasing genome complexity [115]. This increase in genome complexity can confuse the results of genetic analysis as many bioinformatics tools are designed and tested using data from diploid species.

2.2.2 Genotype and Phenotype

The combination of genes an individual inherits determines the individual's genotype. Genes largely control the physical characteristics (phenotype) of the organism as well as control the cellular mechanisms that keep the organism alive. Different phenotypes, such as blonde versus brown hair or purple versus red flowers, result from variations within the genetic code [56, 76]. These variations result in alternative versions of genes known as alleles [97]. Allelic variation results in many traits of interest such as disease resistance, drought and cold tolerance, yield, oil content in plants and genetic disorders and disease susceptibility in humans. Such variations are often the target of research. In order to determine the genetic causes of these phenotypes, molecular markers (discussed in Section 2.2.5) are used to track allelic variants.

2.2.3 Model and non-Model Organisms

A model organism is a species that is used in the study of other, usually more complex, organisms. Model organisms often have small genomes with low genome complexity. They also tend to be easy to work with in the laboratory and as a result have many genomic resources developed for them. These resources often include gene sequences that are well-defined and annotated and finished genome sequences. In comparison, non-model organisms tend to have large, complex genomes that present challenges for the development of genomic resources.

2.2.4 Polymerase Chain Reaction

Polymerase chain reaction (PCR) is a method for generating copies of a target region of a DNA molecule. This process, known as amplification, begins by separating the two strands of the DNA molecule and requires two short DNA fragments known as PCR primers. Each primer is complementary to a single stand of the DNA molecule flanking the target region. The primers bind to the single stranded DNA and an enzyme extends the primers to recreate the double stranded DNA, resulting in two copies. By completing this process several times, the number of copies of the target fragment can be increased greatly [64]. The size of the amplified fragments (product size) can be estimated by the number of base pairs between the two PCR primers.

2.2.5 Molecular Markers

A molecular marker is a DNA fragment, associated with a precise genomic position, that can be developed from DNA variation found among individuals. Markers vary in the complexity of their development and application, as well as their density across the genome. Ideally they are developed to anchor a gene that confers a trait of interest [34, 92, 126, 118]. There are several types of molecular markers, including simple sequence repeats (SSRs) and SNPs. SSRs are short (2–4 base pairs) repetitive DNA sequences (eg. ATATATAT) that can be used as molecular markers by measuring differences in the length of the repeat [91, 118].

2.3 Single Nucleotide Polymorphisms

Compared to other types of molecular markers, SNPs are the most abundant variation in eukaryotic genomes and as such have become the molecular marker of choice. For example, recent evidence suggests that when comparing human DNA from two individuals a SNP is expected on average once every 1000–2000 base pairs [24, 27]. In plants there is evidence of SNPs being even more abundant. For example, in maize (*Zea mays* L.) coding regions (regions which result in the production of a protein), one SNP was found per 124 base pairs and in non-coding regions one SNP per 31 base pairs [18]. There are two types of SNPs, transitions and transversions. Transitions occur when either a purine (adenine or guanine) is converted to a purine (A→G or G→A) or a pyrimidine (cytosine or thymine) is converted to a pyrimidine (C→T or T→C). Transversions occur when either a purine is converted to a pyrimidine (A→C, A→T, G→C, or G→T) or a pyrimidine is converted to a purine (T→A, T→G, C→A, or C→G) [97]. There are many applications for SNPs in both human and plant genetics, some of which will be discussed in Section 2.3.1.

The most common method for finding SNPs is to compare the DNA sequence of two or more closely related individuals; this is often done by sequencing the DNA of the individuals and comparing the results. Even though these variations are common in most genomes, the process of accurately sequencing and characterizing

the variations is complex; this process is generally referred to as SNP discovery.

2.3.1 SNP Applications

The detection of SNPs is an important tool in human, animal, and plant genetics since this natural variation can be utilized for the development of genetic markers that can identify genes causing susceptibility to complex diseases or other traits of interest [4, 79, 7]. Traits of interest in plants might include drought tolerance, oil content, or flowering time and have a large impact on the economics of growing these plants for a variety of uses [36].

The abundance of SNPs in eukaryotic genomes allows for the construction of extremely dense genetic maps (defined below) as one or more SNPs may be found in proximity to almost every gene in the genome. These maps may then be used to develop haplotyping systems for genes or regions of interest [90]. Beyond the use of SNPs for generating genetic maps they can be applied to the integration of genetic and physical maps, association studies [5], conservation genetics [26], and genetic diversity analysis [52].

Haplotyping

A haplotype is a group of SNPs with the same genetic pattern among individuals and that are usually in close physical proximity to each other in the genome (Figure 2.3). Haplotypes are generally identified by comparing the same SNP positions in multiple individuals and grouping each pattern. Often a few key SNPs will be enough to distinguish between all possible haplotypes. Linking of individual SNPs into a haplotype has been shown to provide better resolution in studying complex traits, which may show greater association to the haplotype than to any individual SNP [90, 36].

Genetic Maps and Integration with Physical Maps

A genetic map is a representation of the genome based on linked molecular markers (Figure 2.4), whose order and position is determined by measuring the frequency of exchange of genetic material between homologous chromosomes occurring during sexual reproduction (genetic recombination). Markers positioned closer together indicate low frequencies of genetic recombination and markers that are inferred to be inherited together are grouped together to form a linkage group. Depending on how the SNPs were discovered and the level of genomic information known about the organism, the positional information might be based solely on genetic recombination distances (measured in centimorgans) or could be relative to a physical section of DNA [97]. The high density of SNPs in most genomes makes them ideal for creating dense genetic maps [7, 118]. By scoring several related individuals (determining which individuals have the SNP and which do not) across multiple SNP positions, genetic recombinations can be determined and a map can be generated using a program such as JoinMap [108].

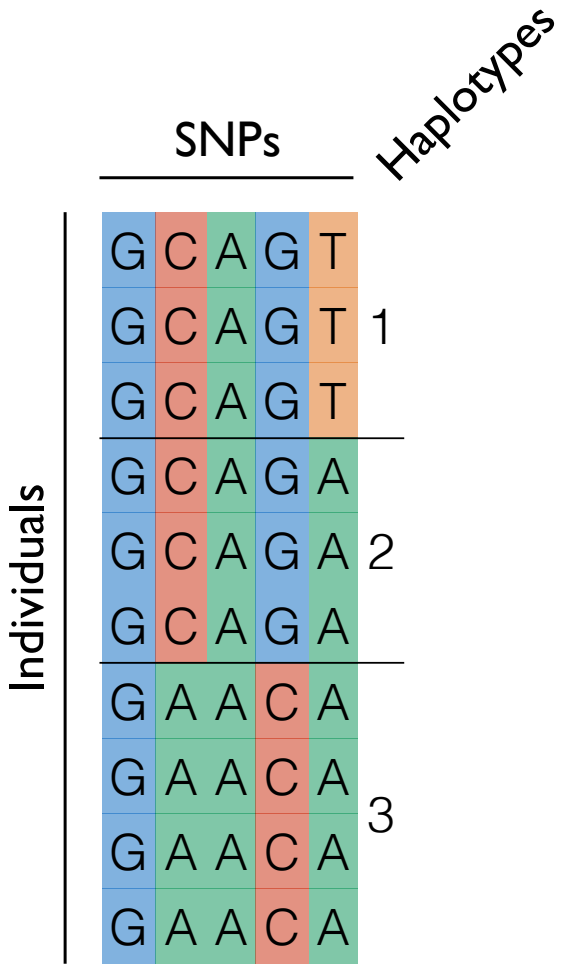

Figure 2.3: Three haplotypes (GCAGT, GCAGA, and GAACA) are distinguished in 10 individuals using 3 SNPs. Each column represents a specific SNP position in the genome while each row represents SNP information from a single individual.

A physical map is a representation of the genome based on large, ordered DNA fragments. Physical maps are generated using a complex process that often uses bacterial artificial chromosomes (BACs). One potential procedure to construct a physical map is as follows. The DNA from a target organism is fragmented into large segments which are inserted into and maintained in BACs (called clones). Whole genome profiling can then be carried out. In whole genome profiling, DNA of individual BAC clones are placed into the wells of a 384 well plate. By collecting a sample of DNA for each BAC clone in a row (24 BACs) or a column (16 BACs), a DNA pool can be created for each row and column of the 384 well plate (Figure 2.5). Pooling of the DNA decreases the number of subsequent sequencing reactions required. DNA from each pool is then fragmented using a restriction enzyme (cuts DNA based on recognition of a specific sequence called a

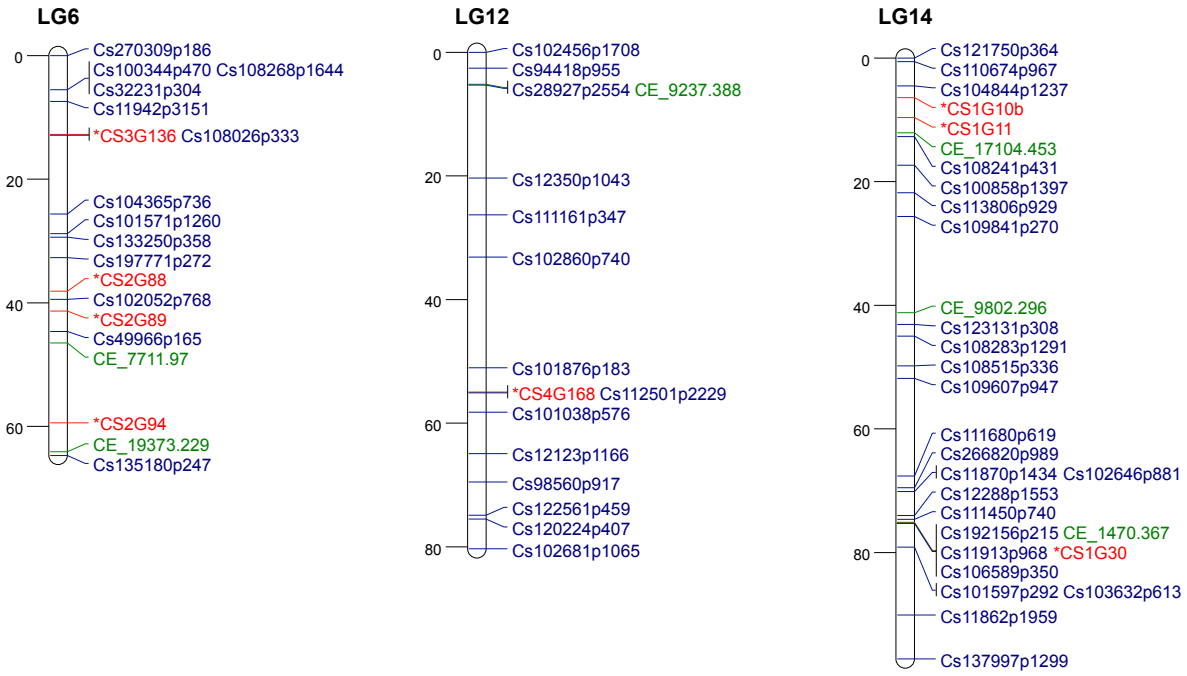

Figure 2.4: A sample of three linkage groups from the genetic map of *Camelina saliva*. Genetic markers appear on the right side of each linkage group, while the numbers on the left side of each linkage group indicate genetic recombination distances in centimorgans.

restriction site). These fragments are then sequenced using next generation sequencing technologies (Section 2.4). Sequencing results can be linked to an individual BAC by finding exact duplicates in both a row pool and column pool (Figure 2.5). After multiple pools are sequenced, the sequencing results can then be used to determine the relative order of the BACs by identifying common fragments that indicate overlapping BAC clones, thus generating a physical map. Since the fragments are of a known size, this map represents the actual base pair length of the genome [117].

The key difference between genetic and physical maps is that genetic maps do not necessarily represent the actual base pair length between markers of the genome while they do for physical maps. Genetic and physical maps can be integrated by locating molecular markers from the genetic map in BAC sequences of the physical map. When one or more markers from the genetic map is found in a BAC, that BAC becomes linked to the genetic map. This is known as integration of the physical and genetic maps [11, 90, 58]. Additionally, genetic maps can be used to anchor results from whole genome sequencing projects in a similar manner [87].

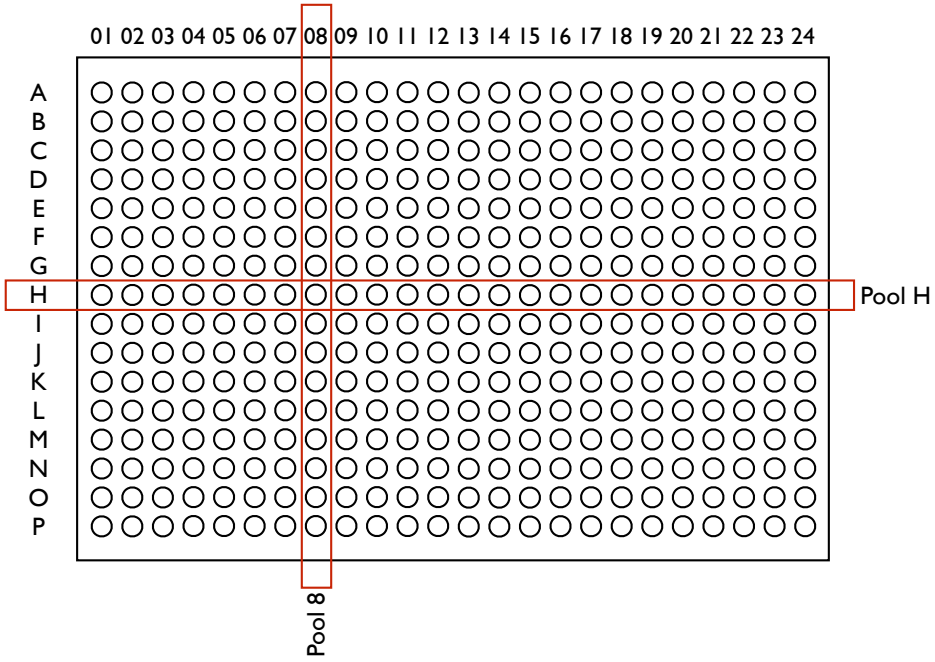

Figure 2.5: Pooling of BAC clones from a 384 well plate. Sequencing results with exact duplicates in the H and 8 pools are determined to be from the BAC clone in the H8 well.

Conservation Genetics and Genetic Diversity

Genetic diversity is an important factor for both conservation of natural species and the improvement of commercial crops. SNPs and other molecular markers have proven to be useful tools in attempts to assess genetic variation in populations. In order to properly assess genetic diversity, it is important to be able to determine variation between individuals. SNPs and SNP haplotypes provide sufficient utility to determine diversity estimates [118, 52].

Genetic diversity is also important for maintaining ecological balances and assessing the relative fitness of populations in the wild. Low genetic variation in a population is a sign of decreased overall fitness. This means that populations have a reduced ability to adapt to changes in their environments which makes them more susceptible to disease. Populations that are reduced to a small number of individuals suffer greatly from reduced genetic variation. Evaluation of genetic variation in populations of interest is an important aspect in effective conservation strategies [52, 59, 118].

Association Studies

Association studies attempt to determine the genes that cause a particular phenotype. Association studies survey molecular markers in multiple, generally unrelated individuals. The number of markers available,

their proximity to each other and their proximity to the genes controlling traits of interest are key factors in the success of association studies. By using high throughput SNP discovery methods, a high density of molecular markers can be obtained. This allows researchers the ability to associate genes with phenotypes where low density genetic maps cannot [36, 126, 90].

SNP Genotyping Arrays

SNP genotyping arrays are mass produced chips containing many probe sequences. A probe sequence is a short DNA sequence that is complementary to a specific region (target) within close proximity (usually less than 100 base pairs) to a known SNP. For the Illumina SNP genotyping arrays used in this thesis, probe sequences are 50 base pairs in length and are used to capture the target in a process called hybridization. Each target then undergoes an extension reaction with labelled nucleotides, which allows for scoring of the target based on the intensity of the signal generated when the labelled nucleotides are scanned [109].

Each probe is fixed to a bead that is placed on a chip for scanning. Custom genotyping arrays offered by Illumina have bead densities which allow for as few as 48 SNPs and as many as one million SNPs to be assayed in as many as 24 individuals for each chip. Individuals are loaded into independent regions of the chip, with each region containing the entire set of SNPs to be assayed [47]. By combining results from multiple chips, the number of individuals assayed can be increased. In order to deal with the large amount of data generated by these arrays, specialized software has been developed to automatically call genotypes and group individuals according to the SNP genotype; this process is referred to as cluster calling [48]. SNP genotyping arrays provide the highest throughput for accurately genotyping large numbers of individuals. Resulting genotype data can then be used across multiple applications, some of which have been described previously.

SNP genotyping array design in polyploid genomes can be complicated by the short length of the probe sequences and the presence of multiple closely related genes. If probe sequences are not carefully designed they may be designed from a sequence common to more than one copy of a gene. These non-specific probe sequences can result in the capture of multiple genome regions during hybridization which can lead to unclear genotyping results during cluster calling [115].

2.3.2 Hemi-SNPs

A major challenge to the application of SNPs in polyploid species is due to the inability to separate reads from the sub-genomes of polyploid species during read mapping. This is particularly problematic in allopolyploid species, which contain not only homologous intra-genomic duplications within each representative sub-genome but also but also contain homoeologous inter-genomic duplicates between the multiple sub-genomes. Hemi-SNPs result from polymorphism between such homoeologous sequences (or between

sub-genomes) within an individual [114]. Figure 2.6 shows the alignment of the sub-genomes (a and b) of two individuals (individual 1 and individual 2) and illustrates how such SNPs are identified during the read mapping process. If the same hemi-SNP variant is identified in both individuals, it represents an inter-homoeologue polymorphism and will not be identified as an allelic difference between the two individuals, thus is effectively monomorphic. However, if only one individual carries the hemi-SNP variant it will be possible to distinguish between the two individuals based on this variation. Hemi-SNPs can be mistaken for simple SNPs due to errors in reference mapping and from insufficient depth of sequencing. In Figure 2.6, for example, if DNA sequencing of individual 1 results in only sampling reads from the (a) sub-genome and DNA sequencing of individual 2 results in reads from the (b) sub-genome, it would appear as though markers can be created which differentiate the individuals. Markers developed based on monomorphic or polymorphic hemi-SNPs effectively assay two independent loci and cannot be specific for a single locus due to the presence of common alleles in both individuals. Due to these complications, SNP discovery in polyploids generally focuses on the detection and use of simple SNPs [114] as they provide more reliable results.

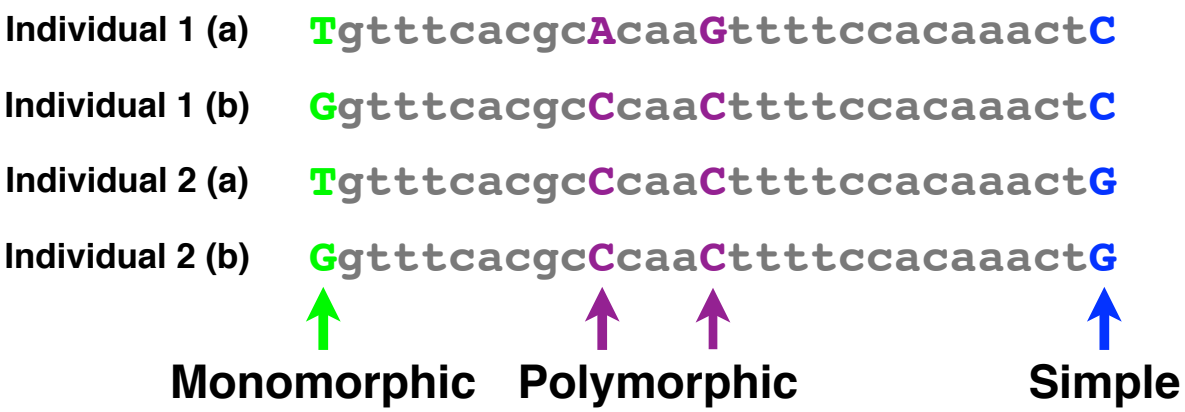

Figure 2.6: Alignment of homoeologous sequences of the sub-genomes (a and b) of two individuals (1 and 2) illustrates the two types of hemi-SNPs (monomorphic and polymorphic), as well as a simple SNP.

2.4 DNA Sequencing Background

DNA sequencing is the method used to determine the order of the nucleotides of a single DNA strand, most commonly by synthesizing the complementary strand and determining which nucleotides have been incorporated. After detection of these complementary incorporated nucleotides, the sequence of the original strand can be determined and output as a series of A's, G's, C's, and T's. The sequence output is typically only a short subsequence of a chromosome and is most commonly referred to as a read.

Often the goal of DNA sequencing is to determine the whole genome of a species, which then can be used as a reference in a variety of further research [60, 15, 121, 116, 87]. To generate a genome sequence, DNA of the target is sequenced and then the reads are *de novo* assembled (Section 2.5.3) into a draft of the genome. Early genomes, such as the human genome, took a long time to sequence and were very expensive. The first human genome was sequenced in 2003 after 12 years and $2.7 billion dollars of research [49]. However, recent advances in DNA sequencing technologies have allowed for much quicker generation of whole genome sequences at a much reduced cost. This has lead to many new species being targeted for whole genome sequencing [15, 121, 116, 87]. As more genomes become available the opportunities for studying genetic variation also increase [90]. Often, studies of genetic variation rely on a technique called re-sequencing, where DNA sequence reads from individuals in the study are mapped (discussed below in Section 2.5.4) to a reference genome sequence [1, 74, 57].

Technological differences in sequencing methods result in differences in read lengths, number of reads, error types, and error rates. Therefore, algorithms used for the analysis of sequence data are often directly dependent on the sequencing methodology employed. While the output formats of the sequencing technologies do vary, the results of DNA sequencing are similar, as sequencers output DNA sequencing reads and an estimated quality value for each nucleotide. Quality values are a measure of the sequencer's confidence in the nucleotide call. This thesis will provide computational tools that process both first generation and next generation sequencing data. For this reason, a description of first generation and next generation sequencing technologies is provided in Section 2.7.

2.5 Bioinformatics and Computational Background

2.5.1 Sequence Alignment

Sequence alignment is a method for determining sequence similarity by comparing two or more DNA, RNA or protein sequences. There are usually two primary results of a sequence alignment: an alignment, which indicates regions of the sequences that are in common, and a score, which indicates how similar the sequences are to each other based on the alignment. An alignment of two sequences can be thought of as a matrix, where the sequences are the rows of the matrix, and the bases of each sequence are placed in order in the columns of the matrix. Although the bases are placed into the columns in order, base pairs are not necessarily adjacent due to the insertion of gaps. Gaps are places in the alignment where a base from some sequences is aligned to no characters of other sequences. Typically, the biological goal of sequence alignment is to predict regions of sequence homology (evolutionary relatedness) by alignment of the homologous regions. Then matches are desired, mismatches could represent mutation, and gaps could represent insertion or deletions in the sequences. The amount of similarity between two sequences is often represented using the alignment

score, which depends on the alignment approach. There are two main approaches to sequence alignment: global alignment and local alignment.

Global Alignment

A global alignment is an alignment of two or more sequences across their entire length. The Needleman-Wunsch Algorithm was developed in 1970 by Needleman and Wunsch for finding the highest scoring alignment between two sequences. Scoring of alignments is usually based on three key components: a match score, a usually positive integer added to the score when aligned nucleotides are the same; a mismatch score, commonly a negative integer or zero added to the score when the aligned nucleotides are not the same; and a gap penalty, commonly a negative integer added to the score when a gap has been inserted in one of the sequences [84]. A common improvement in detecting homology to the Needleman-Wunsch algorithm is to use a more complex gap penalty, such as the affine gap penalty method. The affine gap penalty approach has two components, a gap opening penalty (a usually large negative value added to the score upon the start of a gap) and a gap extension penalty (a much smaller negative value added to the score each time the gap is extended) [128]. Additionally, some algorithms choose to differentiate between terminal gaps (at the ends of sequences) and those found in the middle of sequences [63].

Local Alignment

Local alignment is a method for finding an alignment between subsequences of two or more sequences. The Smith-Waterman Algorithm is a method for finding the highest alignment score between two subsequences and was first characterized by Smith and Waterman in 1981 as a modification of the Needleman-Wunsch global alignment method [105]. The key modification in the local alignment algorithm results in the alignment score never being below zero; this allows for the identification of high scoring subsequences. Figure 2.7 shows a comparison of global and local alignment using a simple scoring structure (match score = +1, mismatch = 0, gap penalty = -1). The local alignment algorithm identifies an alignment that has a score of plus eight, while the global alignment algorithm identifies two alignments both with a score of plus four.

Multiple Sequence Alignment

The goal of multiple sequence alignment is to determine regions of commonality among three or more sequences, therefore it is most common for multiple alignment algorithms to use a variation of the global alignment algorithm. One advantage of multiple alignment compared to pairwise alignment is that multiple alignment can use evidence of similarity in multiple sequences to find small regions of similarity that may not have been recognized using pairwise alignment. Many multiple alignment algorithms are computationally expensive. In fact, the time required to calculate optimal alignments (Needleman-Wunsch,

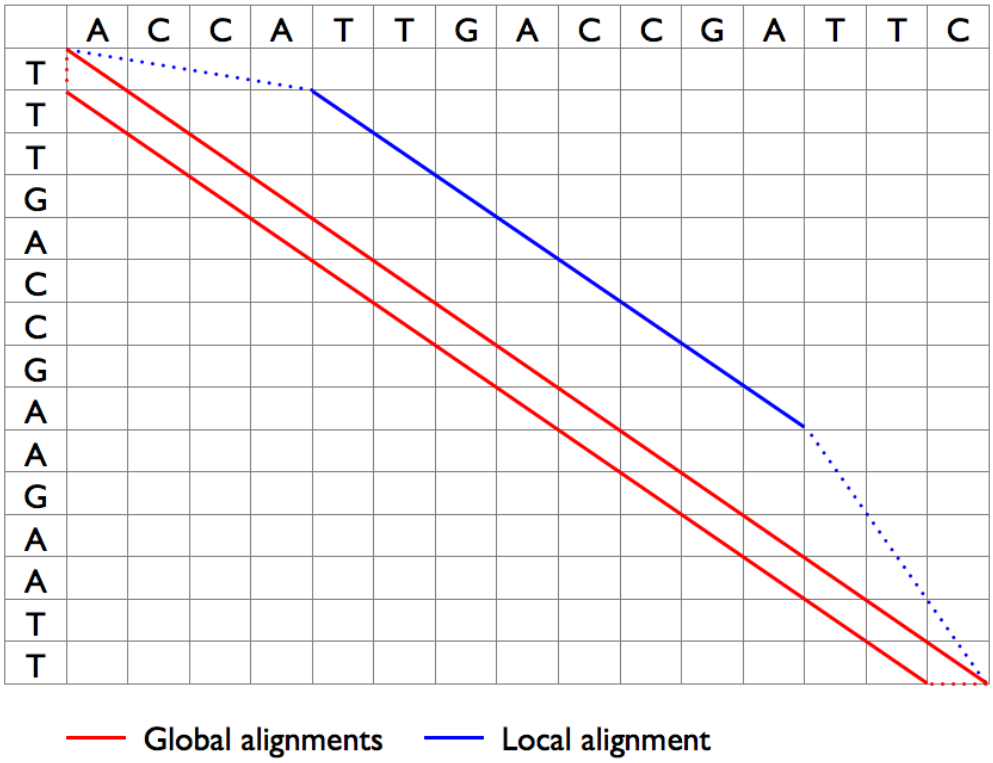

Figure 2.7: Comparison of simple global alignment and simple local alignment (match score = +1, mismatch score = 0, gap penalty = -1) with solid lines representing sequence alignment and dashed lines representing gap insertions. Global alignment compares the strings across their entire length, resulting in more than one alignment with the same score (+4). Local alignment generates a single higher scoring alignment (+8) of only a segment of each sequence.

Smith-Waterman) increases exponentially with the number of sequences aligned. Typically, other heuristic methods are employed, but those can remain difficult as the number of sequences grow [63].

2.5.2 Database Search

Database search is a general term for any algorithm which determines similarity of each sequence in a set of query sequences compared to a database of sequences. Generally, database search algorithms use local alignment based methods to determine sequence similarity [3, 63]. A brief overview of the two methods used in this thesis, BLAST [3] and BLAT [53], are given.

BLAST

BLAST is a tool that searches for sequence similarities between one or more query sequences and the sequences of a DNA or protein database. The BLAST algorithm breaks up each query sequence into short

subsequences and stores them in an index. The database is then scanned using the short sequences to find sequences in the database with similar subsequences. When a match is found, an alignment (called a seed alignment) is generated. If the seed alignment reaches a predetermined score, then extension of the alignment takes place. The alignment can be extended in both directions from the seed. When no further extension or trimming of the alignment increases the score the alignment is determined to be locally optimal. Locally optimal alignments are called high-scoring segment pairs (HSPs). The score of an HSPs alignment is then normalized and the normalized score can then be converted into an expected value known as an E-value. The E-value is the number of distinct HSPs, with at least the normalized score, expected to occur by chance given the size of the search space. The search space is defined as the size (in base pairs) of the query sequence multiplied by the size of the database (in base pairs). The E-value and normalized score are the main factors researchers use in filtering BLAST results for significance [3].

BLAT

BLAT another tool for searching for sequence similarities between query sequences and DNA or protein databases. BLAT is similar to BLAST in that it searches for short matches and extends them into HSPs. However, BLAST requires a specially formatted database while BLAT does not and BLAST generates an index of the query sequences while BLAT generates an index of the database sequences. BLAT also requires perfect or near-perfect seed alignments before extension occurs. Contrary to BLAST which reports alignments of each HSP of a query, BLAT stitches together all HSPs into a single alignment. Additionally, BLAT was developed to align EST and mRNA sequences and therefore is better at detecting splice events [53].

2.5.3 *De novo* Sequence Assembly

De novo sequence assembly is the process of combining short DNA sequencing reads into longer contiguous fragments known as contigs. This is often accomplished by finding overlapping reads and combining them into a single fragment based on the overlapping portion. The overlapping portion of the sequences are combined into a consensus sequence and the number of overlapping reads at a given position is referred to as the read depth. When there are disagreements in the consensus (from either an error in sequencing or in assembly) the most frequent nucleotide at the position is used or an IUPAC ambiguity code can be inserted. IUPAC ambiguity codes (Appendix A) are letters which represent one or more nucleotides, for example the letter R represents either Adenine or Guanine [12]. In whole genome assembly, contigs are often combined using additional read data into scaffolds, which may or may not contain gaps. Scaffolds are then combined into pseudo-molecules representing whole chromosomes by determining the relative order of scaffolds and their orientation to each other [2].

2.5.4 Read Mapping

In comparison to *de novo* assembly, read mapping is essentially a sequence alignment problem where DNA sequenced reads are aligned to a set of previously assembled reference sequences to determine the location of the read within the reference set. As in *de novo* assembly, overlapping reads can be converted into a consensus sequence and the number of reads aligned to a reference position is referred to as the read depth. There are a variety of software packages available for read mapping that fall into three main categories algorithmically. The focus of this section will be to discuss the algorithms and a selection of software packages in more detail.

Local Alignment

Local alignment (Section 2.5.1) has a distinct benefit over global alignment for mapping of reads, as it allows for more errors in the ends of the reads being mapped (where errors are more frequent) [22, 55]. Software packages using local alignment include Mosaik [110], CLC Bio reference assembly [22], and FASTA [88]. An advantage of read mapping using local alignment algorithms is that reads of varying length can all be mapped at the same time, whereas some of the other algorithms (eg. Burrows-Wheeler Transform) require reads of similar length [110, 65, 66].

Index/Seed

As the number of sequences to map became larger it became unfeasible to use the exact alignment methods of local alignment. Heuristic methods were developed that generate an index of either the reads or the reference genome [72, 103]. This index is stored as either a hash or array and is scanned to generate short seed alignments. The purpose of the seed alignments is to narrow the alignment space, reducing overall run time. These seeds are then extended into longer alignments [3, 53, 72]. With the increase in sequencing reads due to next generation sequencing technologies, older index/seed based methods such as BLAST [3] and BLAT [53] were unable to efficiently handle the large amount of data. New methods were developed, such as MAQ [69], SOAP [72], RMAP [103, 102] and several others. These methods usually reduce the number of seed locations by filtering seeds on criteria such as the number of mismatches. Some of these newer methods sacrifice flexibility in terms of input read length in order to perform more quickly. For example, MAQ versions under 0.7.0 have an input read length limit of 63 base pairs, while versions 0.7.0 and above support up to 127 base pairs, and SOAP does not support reads over 60 base pairs. These limitations are generally due to scalability, as when read length increases so does the time and memory footprint to run these algorithms. Increasing from 36 base pair reads to 50 base pair reads triples the running time of SOAP and going from 50 base pairs to 76 base pairs nearly doubles the running time of MAQ and increases the memory footprint by 43% [62].

Burrows-Wheeler Transform

In an attempt to further increase performance of read mapping, the Burrows-Wheeler Transform (BWT) [13] has been adopted by several programs including BWA [65, 66], SOAP2 [73], and Bowtie [62]. BWT is an alternative method for indexing which allows for fast and memory efficient searching. It works by adding a character to the end of a string to be transformed, usually $, which is not in the alphabet of characters to be indexed and is lexicographically less than the characters from the alphabet (Figure 2.8(a)). Then the Burrows-Wheeler matrix for the string is generated by making each row one of the cyclic rotations of the string to be indexed, and the matrix is sorted (Figure 2.8(b) & (c)). The far right column of the matrix is the Burrows-Wheeler Transform of the string (Figure 2.8(c) & (d)) [62, 65]. There are two important properties of the BWT: first, the results are highly compressible and second, the BWT can be converted back to the original string (Figure 2.9) [13]. The compressibility of the BWT results from the tendency of the same characters to group together and the reversibility allows for strings to be stored in their compressed form, retrieved, and then reversed to obtain the original string.

Additionally, the sorted matrix can be used to search for exact matches using an algorithm proposed by Ferragina and Manzini [30]. Algorithms for inexact matching generally use a back-tracking algorithm that first tries to find an exact match and when that fails moves back to a point where a mismatch may be used and the alignment extended from there. To avoid excessive back-tracking the number of mismatches allowed is usually limited [62, 73, 65].

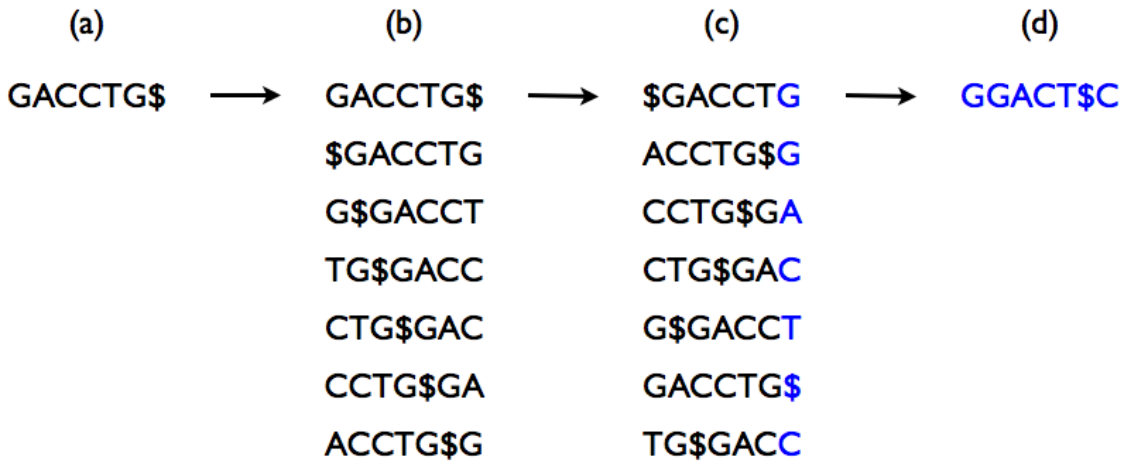

Figure 2.8: The Burrows-Wheeler Transform for the string 'GACCTG' by (a) appending the string terminator character, (b) creating a matrix using the cyclic rotation of the string and (c) sorting that matrix on lexicographical order results in (d) the BWT of 'GACCTG'.

Figure 2.9: Regenerating the original string 'GACCTG' from the BWT sorted matrix using the last first method.

2.5.5 Bit Vectors

A bit vector is a binary string that can be thought of as an array; elements of the bit vector can be accessed directly using an offset and the bit size of each element. In Perl, a common programming language for bioinformatics, the size of an element must be a power of 2, with a minimum size of 2^1 and a maximum size of either 2^{32} or 2^{64} (depending on the underlying architecture of the computer system). Since all of the bits allocated to each element can be used for the input, as opposed to the behaviour of the built-in Perl hash or array structures which have non-mutable element sizes designed around programming flexibility, bit vectors provide a more space efficient data structure. Figure 2.10 gives an example of setting or getting an element from a bit vector and the equivalent action in an array.

Bit Vector	Array
$vec(bit\_numbers, 3, 16) = 8$	$array\_numbers[3] = 8$
$vec(bit\_numbers, 3, 16)$	$array\_numbers[3]$
$vec(bit\_numbers, 3, 16)++$	$array\_numbers[3]++$

Figure 2.10: In Perl the *vec* command allows for bit vector operations similar to array operations. Both the bit vector and the array start at index 0, so to set the 4th element position 3 is used. The *vec* command takes an additional argument, which is the number of bits for each element. In our example 16 bits are allowed for storing each element. The first operation sets the value of the 4th element to 8, the second retrieves the value in the 4th element, and the third increments the value in the 4th element from 8 to 9.

2.5.6 $\mathcal{O}$-notation

An algorithm's time and space complexity is often measured as a function with respect to its input size (or multiple input sizes if there are multiple inputs). Most commonly, these functions are simplified through the use of $\mathcal{O}$-notation. $\mathcal{O}$-notation is a method first used in mathematics to describe the asymptotic upper bound of the growth of a function with respect to its input, ignoring constant factors. For a given function $g(n)$, $\mathcal{O}(g(n))$ describes a set of all functions $(f(n))$ where: there exists positive constants c and n_0, n is sufficiently large $(n > n_0)$, and each function in the set falls between zero and $cg(n)$ $(0 \leq f(n) \leq cg(n))$ [25].

In computer science, $\mathcal{O}$-notation is more often used to classify algorithms, rather than arbitrary functions, based on an upper bound of the algorithm's complexity. For example, an algorithm that takes input of size n and that runs in, at most, time linear to n is described as big-oh of n or more commonly $\mathcal{O}(n)$. $\mathcal{O}$-notation is used to describe the worst-case time or space complexity of an algorithm for all possible inputs, and therefore focuses on the highest-order term. For example, an algorithm with the growth function $n^2 + n$ is $\mathcal{O}(n^2)$. Categorizing algorithms based on their $\mathcal{O}$-notation allows for the comparison of the efficiency of different algorithms and provides an idea of what computational resources may be required for a problem given the size of its input [25].

2.5.7 Ω-notation

Ω-notation (big-omega) is the method used to describe the asymptotic lower bound of the growth of a function with respect to its input (n), for sufficiently large values of n. Similarly to $\mathcal{O}$-notation, Ω-notation ignores constant factors, focusing on the highest-order term. Therefore, for a given function $g(n)$, $\Omega(g(n))$ describes a set of all functions $(f(n))$ where: there exists positive constants c and n_0, n is sufficiently large $(n > n_0)$, and $cg(n)$ falls between zero and each function in the set $(0 \leq cg(n) \leq f(n))$. In computer science it is primarily used to describe the lower bound on the worst-case running time of an algorithm [25].

2.5.8 Θ-notation

Contrarily to Ω-notation and $\mathcal{O}$-notation, Θ-notation is an asymptotically tight bound on the growth of a function, meaning that Θ-notation bounds the growth of a function (from above and below) between two constants for sufficiently large values of n. Therefore, for a given function $g(n)$, $\Theta(g(n))$ describes a set of all functions $(f(n))$ where: there exists positive constants c_1, c_2 and n_0, n is sufficiently large $(n > n_0)$, and each function in the set falls between $c_1 g(n)$ and $c_2 g(n)$ $(0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n))$ [25].

2.5.9 Amdahl's Law

Amdahl's law is a method used in computer science to determine the theoretical maximum speedup of an algorithm with both sequential and parallelized portions. Amdahl's law states that if P is the proportion of an algorithm that can be made parallel and (1-P) is the proportion that cannot be parallelized, then the maximum speedup that can be achieved using N processors is

$$S(N) = \frac{1}{(1-P) + P/N}. \tag{2.1}$$

2.6 SNP Discovery Background

There are several methods for identification of SNPs. However, the most common methods utilize reads sequenced from the target organism's DNA. These reads are then aligned to a reference sequence set (read mapping) and the differences compared. Reference sequences are often obtained from fully sequenced, closely related model organisms, such as humans or rice. However, due to the efficiency and relatively low cost of next generation sequencing technologies, a reference set can now include *de novo* assemblies of more closely related species or the target species itself, which increases the accuracy of SNP discovery. To reduce the likelihood of incorrectly identifying a base change due to sequencing error as a SNP, multiple reads should be considered. The number of reads required to confidently identify a SNP is different for each sequencing method due to the differences in error rate between the sequencing methods [34, 55].

Advances in DNA sequencing technology are having a significant impact on the methods used for SNP discovery. Due to the magnitude of data produced by the next generation sequencers, high-throughput SNP discovery has become easier and more cost effective. However, low-throughput SNP discovery is still the primary approach of researchers targeting a small number of important genes.

2.6.1 Low-throughput SNP Discovery

The most common approach for low-throughput SNP discovery is to use Sanger sequencing to sequence DNA fragments that are captured using PCR primers. These primers are designed to amplify a target region (Figure 2.11), usually containing all or a part of a gene of interest. Fragments are amplified from a diverse set of individuals, sequenced, and the resulting products aligned against each other (Figure 2.12). SNPs are identified based on variations found within the fragment set taking care to distinguish real SNPs from sequencing errors [90]. Due to the quality of the sequenced products, false discovery is generally below 5% [34]. However, due to the targeted nature and the time involved with the laboratory processes associated with this method, it is unsuitable for large-scale SNP discovery projects across an entire genome. For

such projects it is faster and cheaper to utilize next generation sequencing technologies such as those from Roche/454 and Illumina.

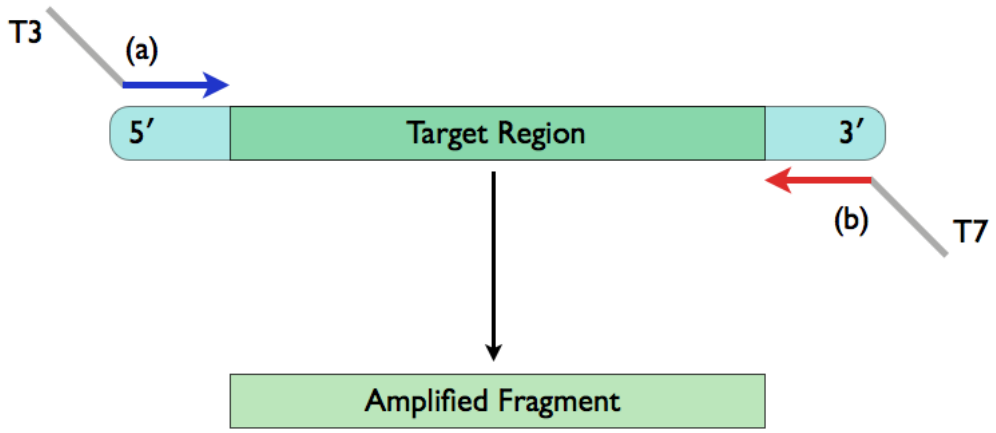

Figure 2.11: Amplification of a target region for sequencing using PCR primers (a, b) results in a fragment that can be sequenced using Sanger sequencing.

2.6.2 High-throughput SNP Discovery

One approach to SNP discovery is to sequence DNA from multiple individuals and compare their sequence to a closely related set of reference sequences. It is important that there is a high level of confidence in the reference sequences as any difference in the DNA of the individual is considered as a potential polymorphism. Once the sequence reads have been aligned to the reference sequence the alignment can be scanned and a report generated of positions where the individual differs from the reference. These potential SNPs can then be compared to a database of known SNPs if available. This comparison allows for the determination of a novel set of SNPs, reduces the need to validate SNPs that are found in the database, and can provide valuable information, such as SNP function. SNPs can also be compared between individuals of a population to determine common characteristics amongst the individuals. SNPs can then be validated using methods such as real-time Polymerase Chain Reaction (rtPCR) or using array based hybridization methods. Validated SNPs can then be provided to researchers for use as reliable molecular markers [34].

2.6.3 SNP Discovery Informatics

There are two main components to the informatics of SNP discovery: read mapping and SNP calling. As previously mentioned, read mapping, the crucial first step of SNP discovery, is the process of aligning the sequenced reads against a reference set of longer more complete sequences, such as those from a *de novo*

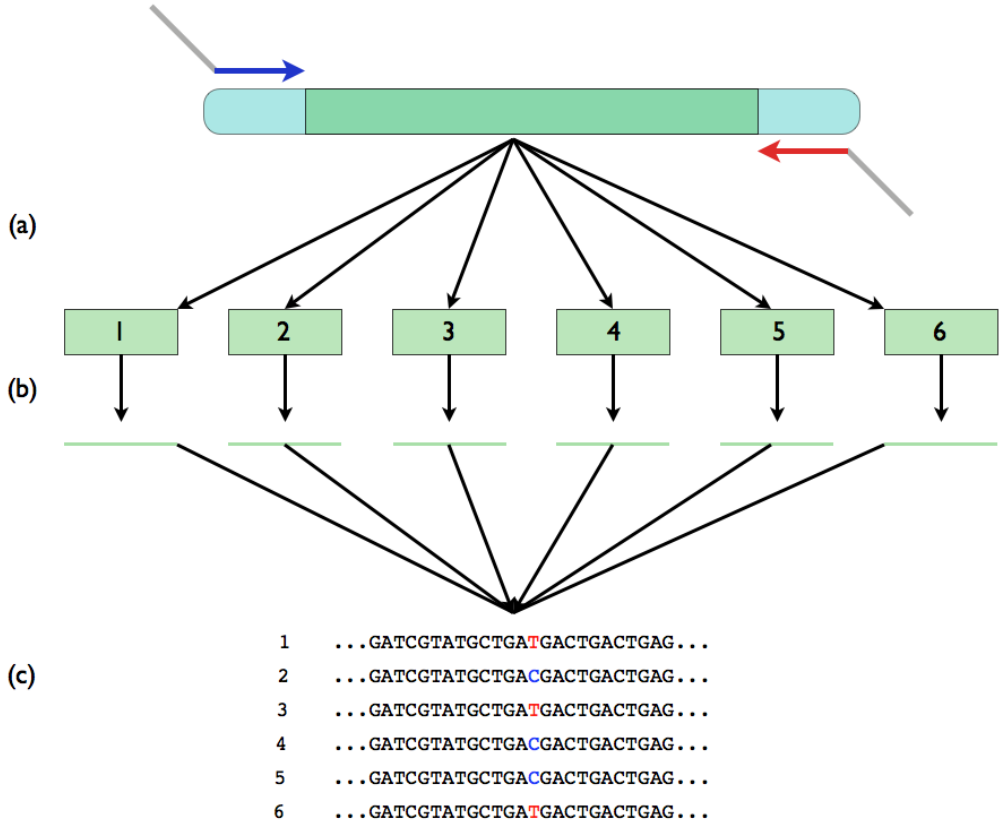

Figure 2.12: Fragments are amplified from multiple individuals (a), then sequenced (b). These sequences can be aligned against one another and checked for polymorphisms (c). An example polymorphic base is shown in colour.

assembly (described in Section 2.5.3). Ideally, these reference sequences are from the same species as the target reads. However, a model or other closely related organism may be used when there is no suitable reference for the species being interrogated.

SNP calling is the process of determining positions within the reference sequence where the sequenced reads differ from that of the reference. This is also often referred to as variant calling or variant detection and is most often done by scanning the alignment for mismatches, determining if a mismatch constitutes a SNP or an error (either in sequencing or alignment), and outputting the SNP calls. Generally, any difference found would be reported as a variant. However, due to errors in sequencing reactions and/or the read mapping, a given variation may not be biologically true. To overcome these errors it is important to have both highly accurate read mappings and to have sufficient read depth providing evidence of the variation. Software such as SOAPSnp [71], CLC Bio's Genomics Workbench and Genomics Server [23], MAQ [69], and SAM Tools [67] all use Bayesian models of statistics to determine if a mismatch qualifies as a SNP. Generally

the output for each SNP called contains several important pieces of information such as the position in the reference sequence, the reference base call, the SNP base call (sometimes referred to as the variant or alternate base call), the total number of reads that are aligned to the reference position (total read depth), and the frequency of the reads containing the SNP (Figure 2.13).

Reference Sequence	Reference position	Reference base	SNP Call	SNP Frequency	Read Depth
Chr 1	100	C	A	100%	20

Figure 2.13: A sample of the information contained in a typical variant call. In this example a variant has been called at base position 100 of chromosome 1. The reference sequence is a "C" and the individual has 20 aligned reads (Read Depth) all (100% Variant Frequency) indicating the variant base"A".

Sequencing and read mapping errors can compound resulting in mismatches that are incorrectly identified as SNPs. Therefore, even after calling SNPs using modern algorithms, it is often necessary to do additional custom screening of the SNP output before utilizing the SNPs in the lab. This is often based on knowledge of the organism, read depth, allele frequency, and quality [73].

2.6.4 Validation of Called SNPs

KASPar

KASPar is a low throughput method that utilizes PCR and fluorescence to reliably detect SNPs. The benefit of this method is that the KASPar primers are easy to develop and validation can be done quickly. However, the downside is that the process is limited to 1536 samples at one time and there is no option for multiplexing to increase the number of SNPs that can be validated. KASPar has been shown to have accuracy of greater than 99.5% as well as high reproducibility [107]. It employs a series of denaturing, annealing and extension steps to ensure that the fluorescent allele specific primer is amplified. Differentially labelled primers result in allele specific clusters that can then be analyzed to determine if the SNP is monomorphic or polymorphic [106]. By performing a KASPar reaction utilizing the reference DNA as well as the DNA from the same sample as the SNP was called, it can be determined if the called SNP was valid.

Arrays

Custom genotyping arrays such as those offered by Illumina allow for a higher throughput validation than that of the KASPar method. Using the highest density arrays, as many as one million SNPs can be validated

at once. By including both the reference DNA sample and a sample from which the SNP was called, it can be determined if the SNP called was likely valid.

2.7 DNA Sequencing Technologies

Knowledge of DNA sequencing technologies provides a better understanding of the challenges faced during the analysis of DNA sequence data for research purposes. As such, each sequencing technology used in this thesis is described below in its own subsection. As the majority of the data analyzed for the thesis work is from next generation sequencers, a summary of their read lengths, output sizes, and error types is given in Table 2.1. The features in this table are the most important factors from a data analysis perspective, to be used for new algorithms developed for this thesis.

2.7.1 First Generation Sequencing Technology

First generation sequencing technologies are those based upon the chain-terminating method developed in 1977 by Frederick Sanger and is widely known as Sanger sequencing. The first step is to generate a large enough quantity of the DNA to be sequenced either using purification or amplification by PCR. A sequencing primer is then used to initiate reverse strand synthesis of each copy of the original target sequence using a mixture of unmodified and modified nucleotides. The modified nucleotides have a fluorescent label attached to them as well as structural changes which block the extension reaction, resulting in fragments of different lengths and thus different molecular weights (Figure 2.14) . The fragments can then be sorted by mass using capillary electrophoresis and the label attached to the termination point read to determine the DNA sequence [98, 104, 112]. First generation sequencers were able to process 96 or 384 samples at a time, producing long read lengths (600–1000bp) with very high accuracy (an estimated one error in 10,000–100,000 sequenced bases) [55]. State-of-the-art sequencers, such as the Applied Biosystems (acquired by Life Technologies) 3730XL, use 96 capillaries and can accept samples in either a 96 well plate or a 384 well plate. These sequencers are capable of completing approximately 12 runs of 96 samples each day resulting in 1152 reads. With an average length of 800 base pairs for each read, the throughput of the 3730XL is 921.6 kilobases per day [6].

2.7.2 Next Generation Sequencing Technologies

Next generation sequencers are considered, for the most part, those based on sequencing by synthesis (SBS). SBS is a process in which the sequence of DNA is read while synthesizing the complementary strand of the DNA sequence. An enzyme known as DNA polymerase, which is responsible for incorporating new bases into the complementary strand during normal DNA synthesis, is used to incorporate nucleotides that can

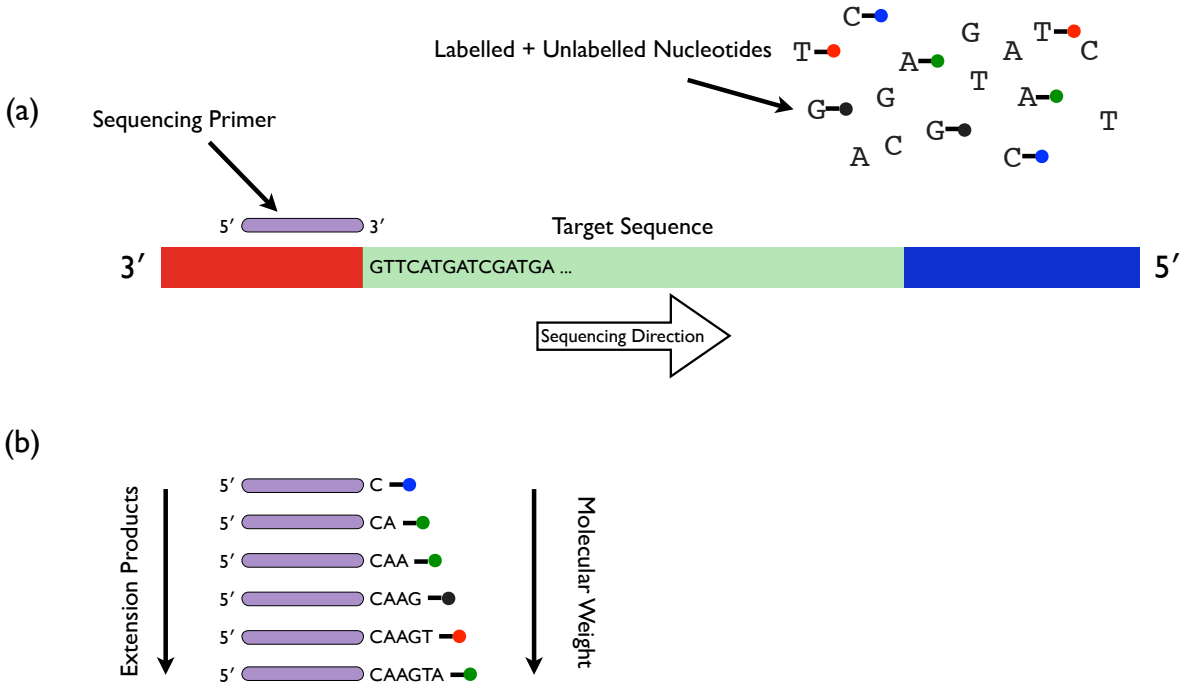

Figure 2.14: Starting point for the Sanger sequencing reaction (a) and a sample of the resulting products of that reaction (b).

then be recorded producing the DNA sequence as they are added to the strand.

Pyrosequencing is a SBS method in which the incorporation of a nucleotide releases a pyrophosphate molecule. The pyrophosphate molecule combines with enzymes and other chemicals in the reaction to produce light. A single nucleotide (A, T, C, or G) is available for incorporation at a time and thus the number of nucleotides incorporated can be determined by the intensity of the light produced. Alternatively, the SBS method used by Illumina is a reversible terminator based method similar to Sanger sequencing, which differs from pyrosequencing in that a fluorescent label is attached to each nucleotide and imaged to determine which base was incorporated [55].

Next generation sequencing technologies usually produce shorter read lengths than those achieved using Sanger sequencing. However, they produce many more reads and thus more overall bases per run. This allows for increased depth (the number of reads per nucleotide of the target organism) of sequencing in comparison to Sanger sequencing at a much lower cost [55]. Depth of sequencing increases the confidence in a base call by providing multiple sources of evidence that a nucleotide has been sequenced without error. Additionally, next generation sequencing methods allow for sampling a larger portion of the target organism's genome [8, 122].

While this is not meant to be a survey of all the different sequencing methods, pertinent information is

included for the sequencers from which data will be examined in this thesis (Roche/454 GS FLX Titanium, Illumina GA_{IIx}, Illumina HiSeq 2000). Also included is information on several newer sequencers that are currently being released (Table 2.1). This includes two lower cost, bench-top style sequencers (Roche/454 GS Junior and Illumina MiSeq) designed to provide lower throughput for researchers either without access to a larger sequencing centre or who do not require the higher volume of data that can be generated by the larger machines. These machines are of interest as they will make sequencing more accessible to every lab, which in turn will create high demand for good software that does not require dedicated staff to perform the data analysis.

Sequencer	Read Length (bp)	Output		Run Time	Error Type
		Base pairs (bp)	Disc Space		
Roche/454 GS FLX Titanium	400	400–600 million	3.1–4.8 GB	10 hours	Indel
Roche/454 GS Junior	400	35 million	0.28 GB	12 hours	Indel
Illumina GA_{IIx}	1x35, 2x50, 2x75, 2x100, 2x150	10–95 billion	24–230 GB	2–14 days	Substitution
Illumina HiSeq 1000/2000	1x35, 2x50, 2x100	47–300 billion	114–726 GB	1.5–8.5 days	Substitution
Illumina HiSeq 1500/2500	1x35, 2x50, 2x100	95–600 billion	0.225–1.42 TB	2–11 days	Substitution
Illumina MiSeq	1x35, 2x25, 2x100, 2x150, 2x250	0.54–8.5 billion	1.3–20.6 GB	4–39 hours	Substitution

Table 2.1: Summary of next generation sequencers including read length, throughput per run, output size, run time and most common error type. Illumina single reads are labelled as 1x, while 2x indicates two sequences reads from a single template (one from each end). Disc space estimates are based on 8.3 bytes per base pair for the Roche/454 platforms and 2.6 bytes per base pair for the Illumina platforms; these estimates were calculated using uncompressed data for each platform. Error types are either insertion/deletion errors (Indel) or substitution errors. Data current as of January 2014.

Roche/454

The Roche/454 machines use the pyrosequencing methodology. Pyrosequencing involves four major steps: generation of a single-stranded template library, emulsion based PCR for clonal amplification, sequence data generation by sequencing by synthesis, and data analysis [94]. The average read length of a pyrosequencing run is 400 base pairs, which is substantially longer than the other next generation sequencing methods [95, 44]. However, this technique has problems resolving homopolymers (stretches of a single base) greater than 8 due to saturation of the sensor that detects the amount of light given off during an incorporation and has lower throughput in comparison to other next generation sequencers [77, 95, 44].

The first stage of pyrosequencing involves the creation of a single-stranded template library. Library production can be performed on the following materials: complementary DNA (cDNA), genomic DNA (gDNA), Polymerase Chain Reaction (PCR) products, and Bacterial Artificial Chromosomes (BACs). Starting materials larger than 800 bases must be fragmented before proceeding with the remainder of the library preparation; this fragmentation occurs at random and produces 300-800 base pair fragments.

Double stranded DNA is denatured into single stranded DNA and sequencing adapters are attached to both the $3'$ and $5'$ ends of each fragment (Figure 2.15a) [94]. The randomness of the library preparation is examined and biases discovered [122].

The second stage of pyrosequencing involves the amplification of individual single-stranded DNA fragments. This is done using a process called emulsion PCR (emPCR). Individual DNA fragments are captured on separate beads by using an excess of beads to fragments. These beads are captured in a droplet of PCR reagents in oil called a microreactor. The microreactors allow for separate DNA amplification resulting in each bead containing several million copies of an individual fragment (Figure 2.15b) [77].

The third stage of pyrosequencing is the generation of sequence data using sequencing by synthesis. Beads are deposited into wells of a fibre-optic plate using centrifugation (Figure 2.16a). The size of the beads and the wells are matched to allow only 1 bead per well. However, in practise multiple beads are observed in approximately 2-5% of wells that contain beads. The plate is then flooded with smaller enzyme beads (Figure 2.16b) that serve two purposes. The first is to immobilize the beads carrying the template strand to keep them from washing out of the well as the nucleotides flow across the surface of the plate. The second is that these enzyme beads have adenosine tri-phosphate (ATP) sulphurylase, luciferase, and other enzymes bound to their surface; these enzymes are required for the chemical reaction which leads to the emission of light during the sequencing process.

During a flow cycle, each nucleotide is iteratively flowed over the plate in a pre-determined order (T,A,C,G) with a wash cycle in-between to remove unincorporated nucleotides. If a nucleotide is complementary to the template strand DNA polymerase present in the well extends the complementary strand by that nucleotide (Figure 2.17a,b). Since there is an abundance of free nucleotides and the reaction is not

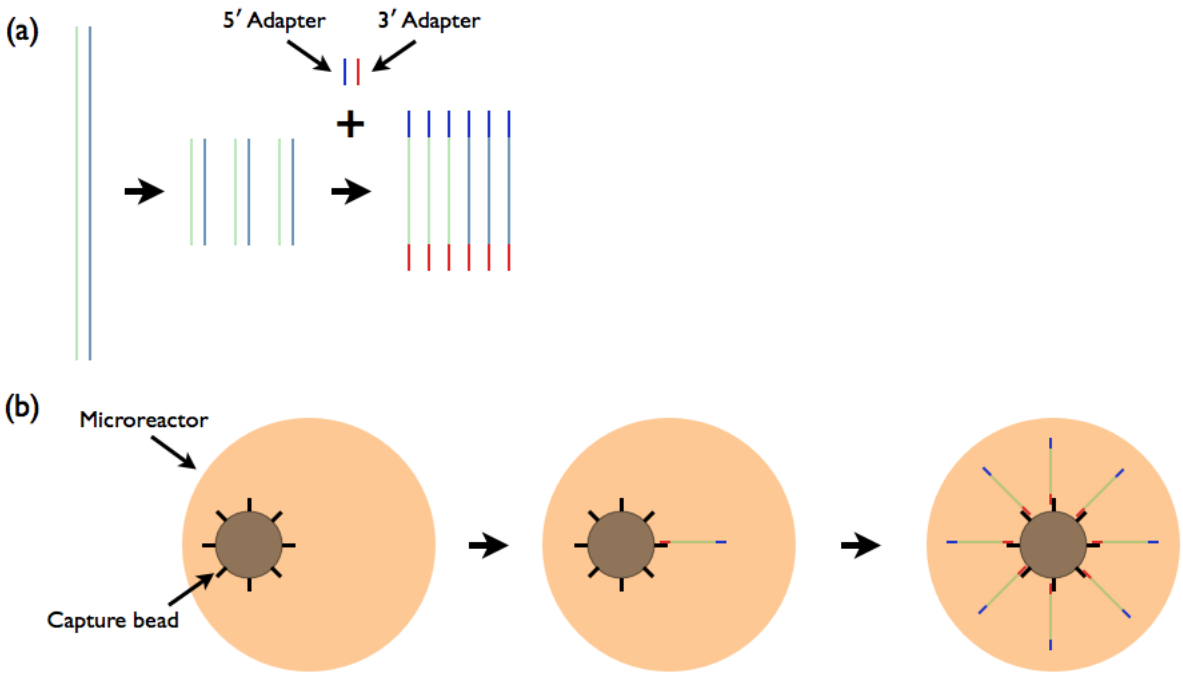

Figure 2.15: Double stranded template sequences are sheared to an appropriate size, the fragments are denatured into single strands and adapters annealed to the ends (a). A single bead is encompassed in a drop of oil forming a microreactor. Excess of beads to fragments results in one fragment per bead. This fragment is then amplified using PCR to create a bead with multiple copies attached to it (b).

halted by an incorporation event, as in the Illumina method, homopolymer stretches are incorporated at the same time (Figure 2.17c). Addition of one or more nucleotides results in the release of a free pyrophosphate (PP_i) that reacts with free adenylyl sulfate (APS) and the ATP sulphurylase bound to the enzyme beads to create ATP. The ATP and luciferase bound to the enzyme bead interact in a chemical reaction which leads to the emission of light. Light is transmitted through the bottom of the fibre-optic plate and is captured by approximately 9 pixels of the charge-coupled device (CCD) sensor. The light intensities that are captured by each of the 9 pixels covering a well are summed to produce a signal for that well during the particular nucleotide flow in which the light was captured [94, 55, 77].

The fourth and final stage of pyrosequencing is the processing of the imaging data. Although each well of the plate only requires 9 pixels, the large number of wells results in 32 megabytes of data for each image captured of the entire plate. In order to process the image data in real time, the control computer uses a Field Programmable Gate Array (FPGA) containing 6 million gates. Wells with target sequence are determined by the detection of a 4 nucleotide key sequence during the first two flow cycles. During the remaining flow cycles the raw signal intensity of the incorporated nucleotide has background signal subtracted, is normalized, and converted to the number of incorporated bases producing a flowgram (Figure 2.18). The read encoded by

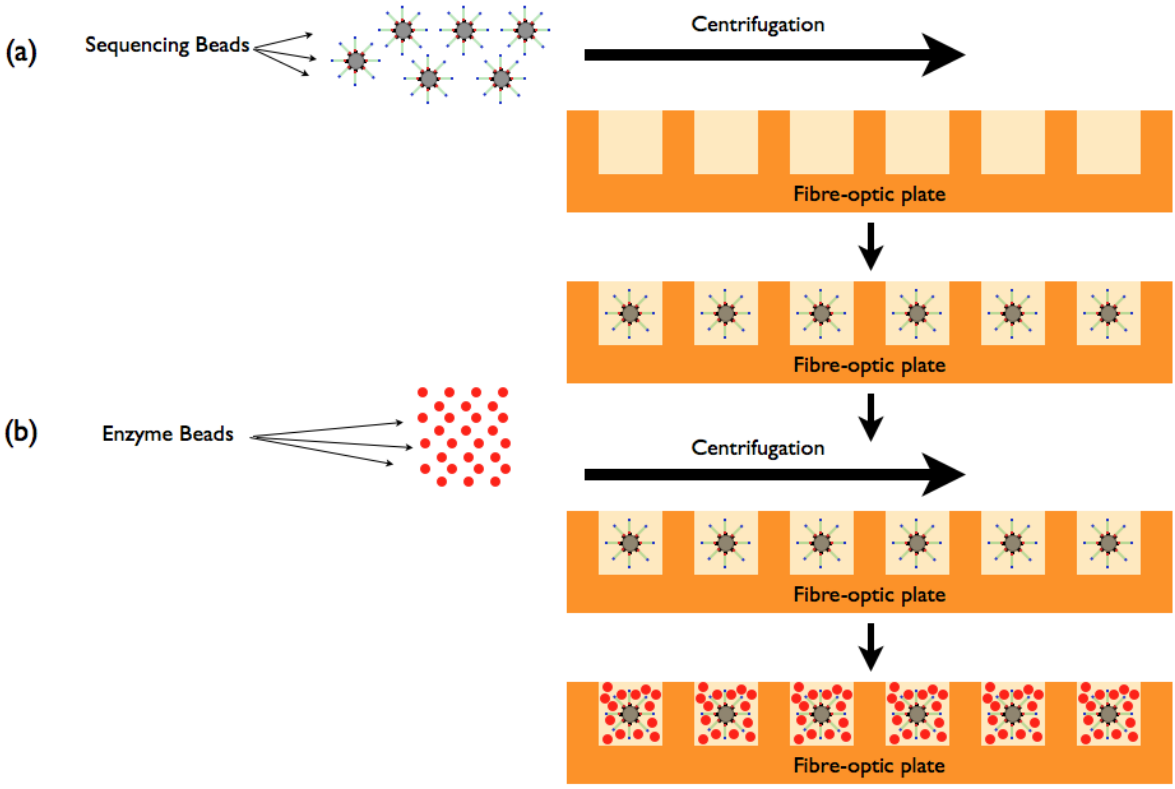

Figure 2.16: Beads containing amplified sequences are deposited in the wells of the fibre-optic plate using centrifugation (a). The smaller enzyme beads are added next using the same method (b).

the flowgram will be reverse complemented to give the 5′ to 3′ sequence of the target sequence. The output from the pyrosequencing run contains the flowgram, the normalized signal, and base calls with quality values. Quality values are extracted from the flowgram using algorithms to detect common errors such as incomplete extension (e.g. incorporating only 4 bases where 5 should be) and carry forward (e.g. incorporation of a nucleotide out of order due to nucleotides trapped in the well) [77]. The most common error on the 454 platform are insertion or deletion (indel) errors. These errors usually result from mistakes in detecting the number of nucleotides incorporated during homopolymers.

Genome Sequencer (GS) FLX Titanium The GS FLX Titanium is the flagship sequencer from Roche/454 and offers a throughput of 400–600 million bases per 10 hour run. Roche claims accuracy greater than 99% for bases 1–399 and an accuracy of 99% at 400 bp. In order to process the flowgram files into base and quality information a computing cluster is recommended [95].

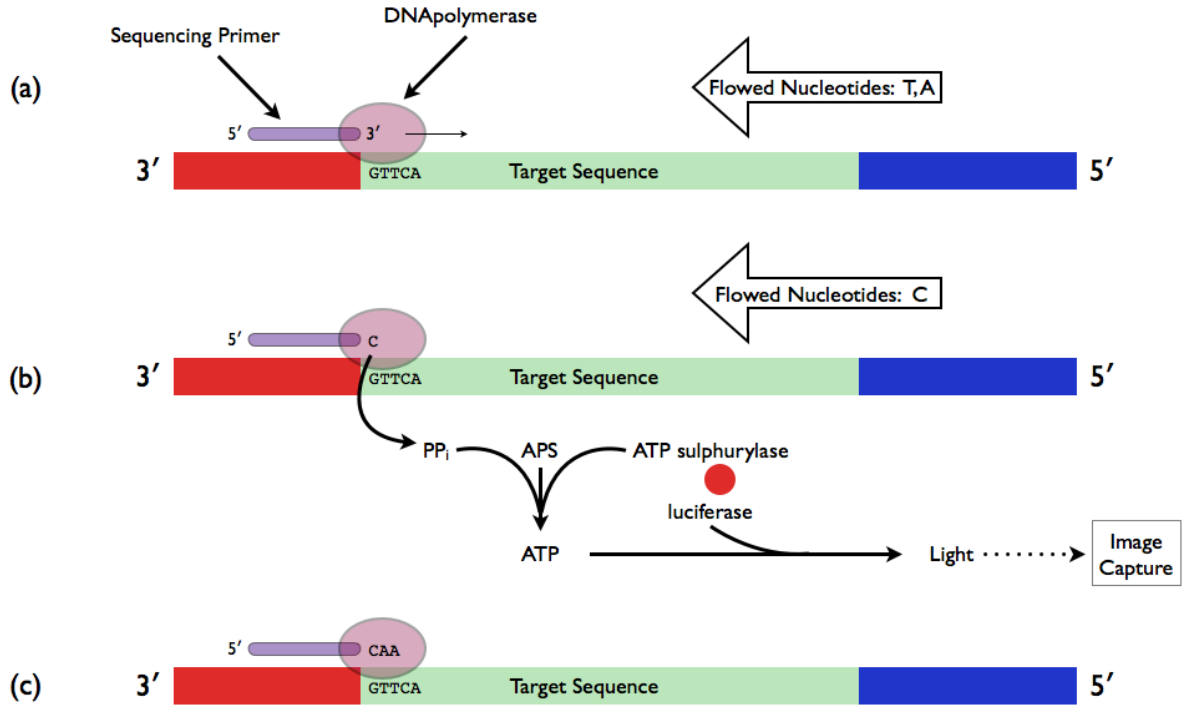

Figure 2.17: When the complementary strand does not match the nucleotide being flowed across the plate no incorporation event occurs (a). When the flowed nucleotide is complementary a chain reaction takes place starting with the incorporation of the nucleotide. This results in a free pyrophosphate being converted into light (b). Stretches of a single nucleotide are incorporated and captured at the same time resulting in a higher light signal (c).

GS Junior The GS Junior is a bench-top version of the FLX Titanium system designed to offer reduced throughput but at a more accessible price level (approximately \$150–200K vs \$400K). It provides the same average read lengths and accuracy as the FLX Titanium with a throughput of approximately 35 million bases instead of 400–600 million bases in a 12 hour run time. This machine requires only a desktop computer for processing the image data [93].

Illumina

Similarly to the Roche/454 method, the Illumina sequencing method has four major steps: generation of the sequencing library, clonal amplification of individual reads, SBS, and data analysis [10]. The Illumina method produces read lengths of 250 bases or less, yet due to the high number of reads that can be extracted from a single flow cell (the glass substrate to which template sequences are bound) produces more data per run than the Roche/454 method [95, 42, 44].

In the first stage the sequencing template library is generated by creating DNA fragments of the ap-

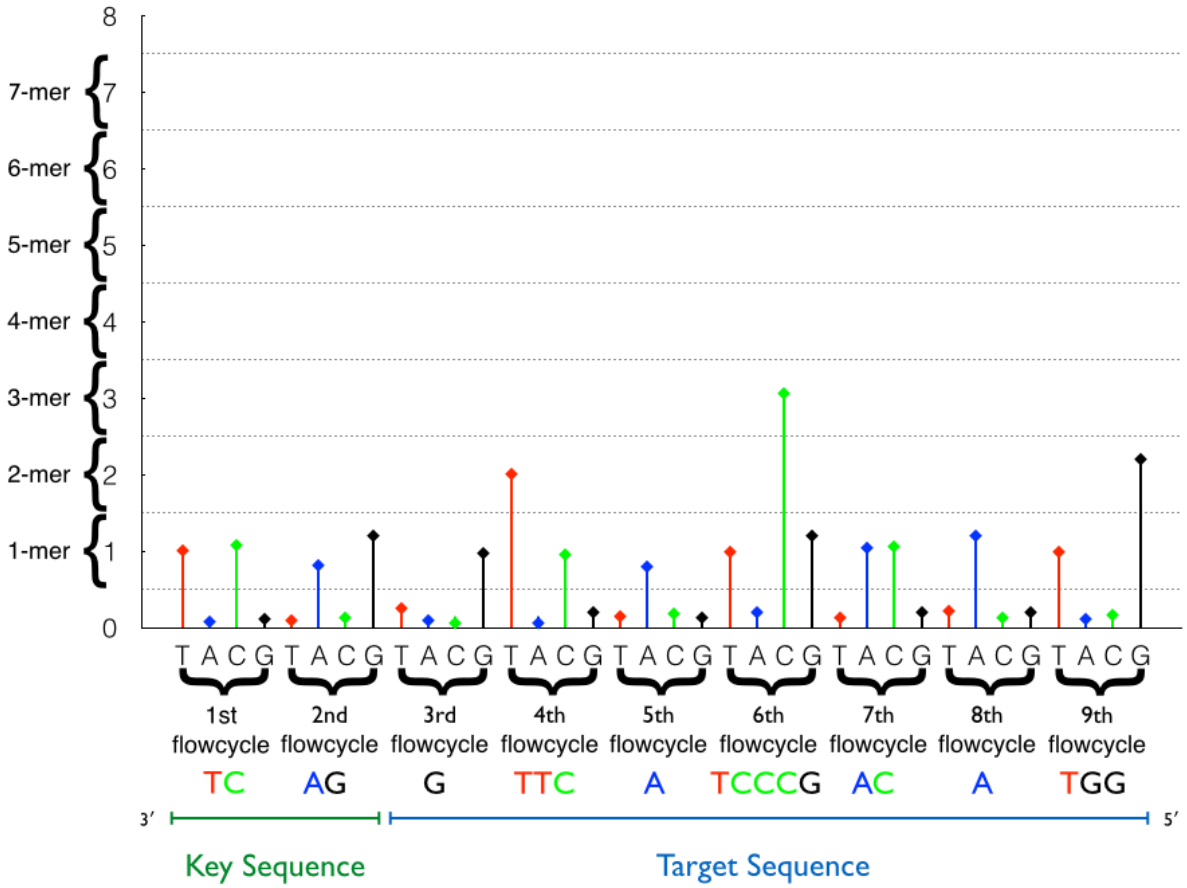

Figure 2.18: A sample of a flowgram (nucleotide flow order T,C,A,G) showing the first nine flow cycles of a read. The first and second flow cycles read the 4 nucleotide key sequence and allows for signal calibration. Flow cycles 4 through 9 result in a 16 nucleotide read (GTTCATCCCGACATGG).

propriate size, usually by shearing of longer fragments. The fragment size for the library is determined by the read length being sequenced and the insert size of the library (paired-end only). These fragments are joined to a pair of short adapter sequences (one on each end of the fragment) and then denatured into single strands (Figure 2.19a) [10]. These adapters allow the sequence to be bound to the surface of the flow cell for sequencing. The sequences are flowed at a low concentration across the flow cell and the adapters bind to complementary adapter sequences that have been fixed on the flow cell [55]. The complementary strand of the template is then synthesized starting with the adapter sequence fixed to the flow cell. The original strand can then be removed and the second stage begun (Figure 2.19b) [10].

In the second stage of Illumina SBS the single template sequences bound to the flow cell are amplified into clusters using a process called 'bridging' amplification (Figure 2.20). After the fragments have been fixed to the flow cell the adapter sequence on the free end binds to the complementary sequence on the flow

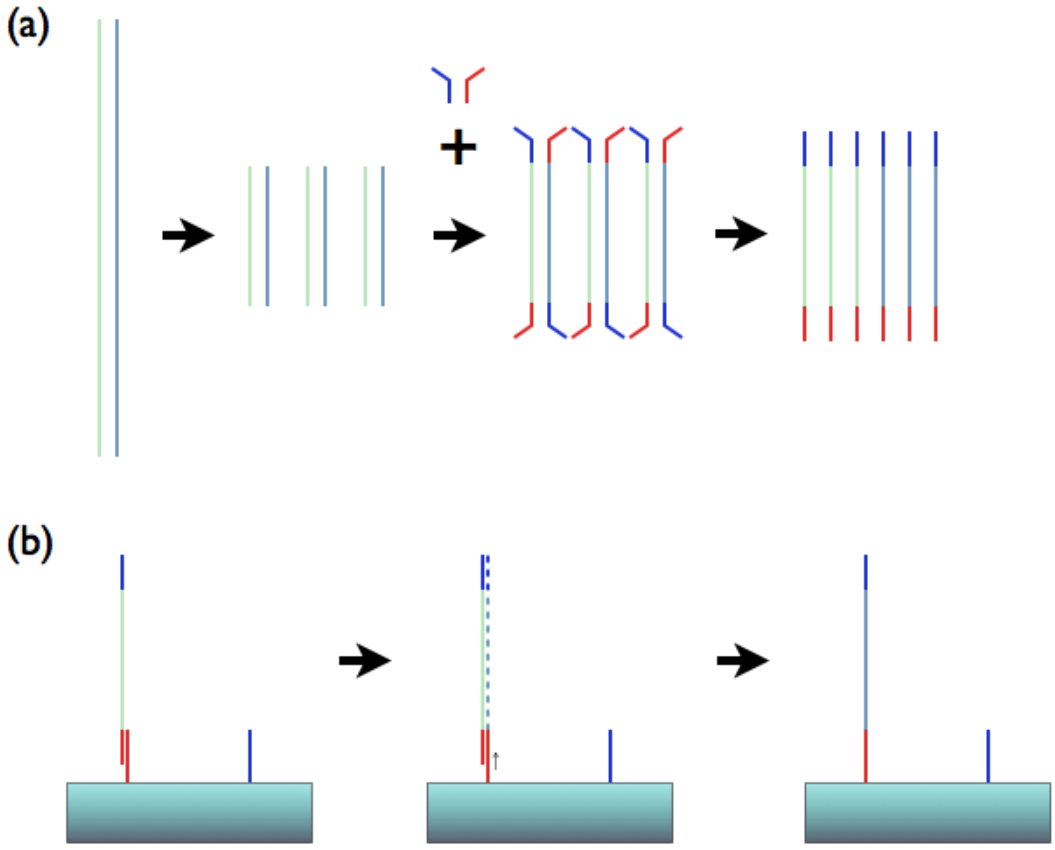

Figure 2.19: Double stranded template sequences are sheared to an appropriate size, adapters are annealed to the ends and the fragments denatured into single strands (a). Single stranded sequences are bound to complementary adapter sequences fixed to the flow cell and the complementary sequence (dashed) synthesized before removal of the original template, leaving only the fixed sequences for sequencing (b).

cell creating a bridge; this bridge can then be used to form the second strand. Once the second strand has been synthesized the bridge is denatured resulting in two complementary copies of the same sequence. This process of binding, copying, and denaturing is repeated for several cycles yielding a cluster of approximately 1 µm. This cluster contains reads in both the forward and reverse directions so in order to have a cohesive cluster one strand is removed before sequencing, by cleaving the sequence using a cleavage site built into the small fragment fixed to the array [10, 55].

The third stage is SBS; instead of detecting incorporation events using light released during incorporation as the 454 does, Illumina uses a reversible terminator chemistry. First, a sequencing primer is bound to the adapter on the free end of each read in a cluster. Using a specially designed DNA polymerase, fluorescent dye labelled nucleotides are incorporated. The Illumina reversible terminator chemistry is similar to the

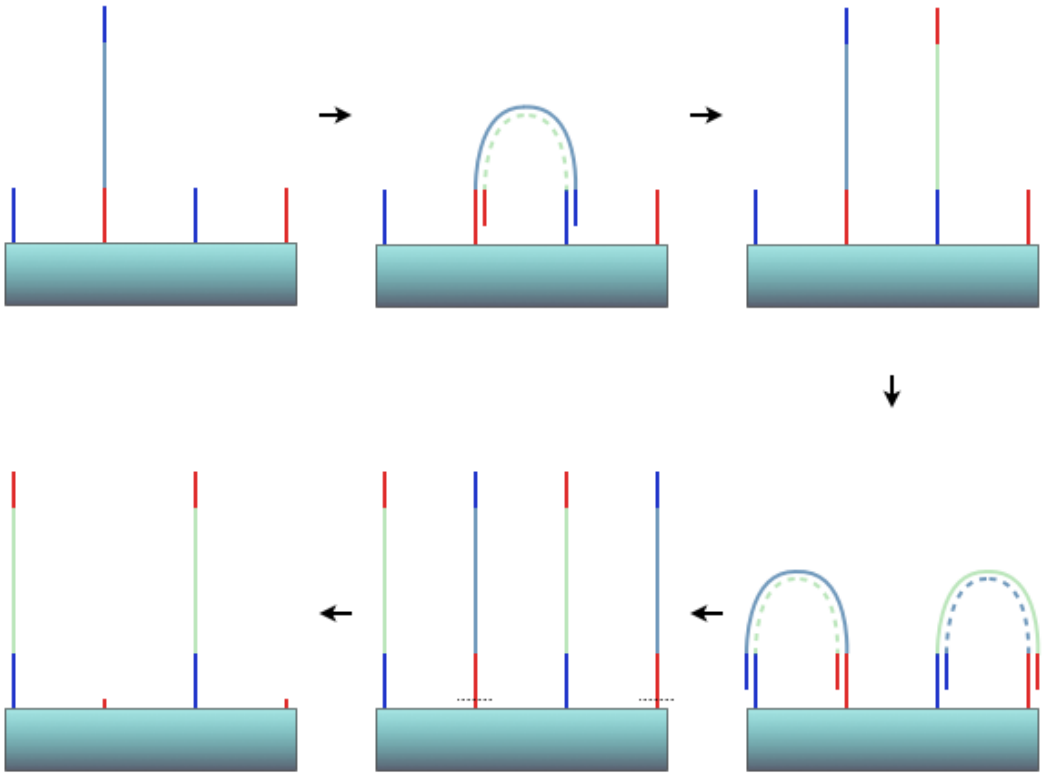

Figure 2.20: Illumina 'Bridging' Amplification involves a series of annealing, extension, and denaturation cycles to amplify the target sequence in each cluster. Since this process amplifies both the forward (light green) and reverse (light blue) directions a step is required to remove one direction by cleaving the oligonucleotide that is fixed to the flow cell.

Sanger method of sequencing in that the incorporation of bases is stopped by the addition of a nucleotide, the incorporated base is then read by exciting the fluorescent dye affixed to the nucleotide with a laser. The fluorescence of each sequence is captured in an image and the fluorescent labels are removed so that another incorporation event can take place (Figure 2.21). The terminator chemistry allows all four nucleotides to be flowed over the cell at the same time, reducing misincorporation events. Four fluorescent dyes are used, one for each nucleotide, with two excited using a red laser (A/C) and two excited using a green laser (T/G). To distinguish between the two nucleotides excited by the same laser optical filters are employed. Therefore, four images are taken at each imaging step (one for each filter) [10, 55]. Since an image is taken after each incorporation there is less risk of misreading the number of bases in a homopolymer in comparison to the pyrosequencing method. However, since two nucleotides are read using the same laser excitation, substitution errors are generally higher with the Illumina platform.

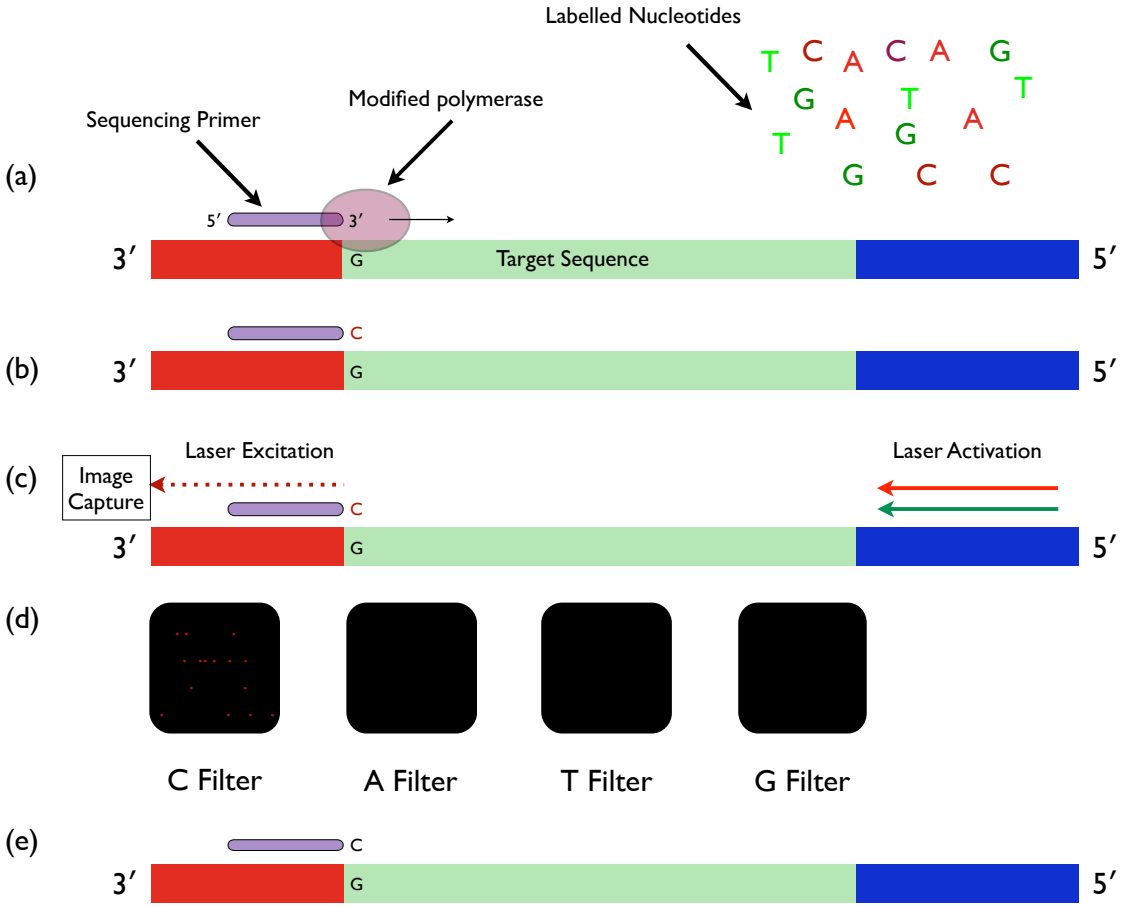

Figure 2.21: Initial setup of the sequencing reaction showing the sequencing primer, the polymerase modified for incorporating labelled nucleotides, and the pool of labelled nucleotides (a). Incorporation of the labelled complementary base into the sequence (b). Activation of the laser causing excitation of the fluorescent dye and the subsequent image capture of the event (c). Simulated imaging results (d) from each filter for the incorporation event in (b). Removal of the fluorescent dye allows the reaction to continue (e).

Genome Analyzer $_{\texttt{IIx}}$ The Genome Analyzer (GA) $_{\texttt{IIx}}$ was the flagship sequencer from Illumina before the HiSeq 1000/2000 sequencers were released. The $\text{GA}_{\texttt{IIx}}$ is capable of producing 10-95 gigabases per 2-14 day run depending on the read length chosen. Shorter read lengths result in quicker run times and lower throughput. As with all sequencers the accuracy tends to decrease towards the end of the read. Illumina claims that greater than 85% of reads have an accuracy of 99.9% for the 2x50 bp read length and greater than 80% of reads have an accuracy of 99.9% for the 2x100 bp read length. The $\text{GA}_{\texttt{IIx}}$ can produce reads of 1x35 bp, 2x50 bp, 2x75 bp, 2x100 bp, and 2x150 bp. In order to capture this data and process it into sequencing reads a dedicated cluster is recommended [42].

HiSeq The HiSeq series are Illumina's new flagship sequencers. They boast higher throughput and shorter run times than that of the $\text{GA}_{\texttt{IIx}}$ with no reduction in accuracy. The HiSeq 1000/2000 will produce between 47 and 300 gigabases per run (1.5 day and 8.5 day runs, respectively) and the HiSeq 1500/2500 will produce 95 to 600 gigabases per run (2 and 11 day runs, respectively). The HiSeq systems can produce read lengths of 1x35 bp, 2x50 bp, and 2x100 bp. In order to facilitate the data collection a cluster is required for both of these machines [43, 44]. Recently Illumina has also developed the Illumina BaseSpace, a cloud storage and computing environment available for the HiSeq and MiSeq. This environment allows the user to store up to 1 Terabyte of run data for free and provides online access to many bioinformatics tools for data analysis. It also is designed to allow easy online collaboration for data analysis [46].

MiSeq The MiSeq is Illumina's new low throughput sequencer and is designed to offer sequencing capabilities to smaller labs. It will produce between 540 megabases and 8.5 gigabases of data per run (4 and 39 hours, respectively). It can produce reads of 1x35 bp, 2x25 bp, 2x100 bp, and 2x150 bp and Illumina claims that it achieves accuracy of 99.9% for greater than 90% of the 35 bp reads, greater than 80% of the 2x100 bp reads, and greater than 75% of the 2x150bp reads. The MiSeq does not require a cluster for data collection as it has a built in computing component as well as access to Illumina BaseSpace [45, 46].

2.8 DNA Sequencing Methodologies

The cost of DNA sequencing can become prohibitive based on factors such as desired read depth of sequencing, size of the genome to sequence, and the sequencing platform being used. For this reason, different sequencing methodologies have been developed to allow flexibility in the coverage of genome sequencing. The two methods represented in this thesis, shotgun sequencing and reduced representation sequencing, are discussed below in their own subsections.

2.8.1 Shotgun

The shotgun approach to DNA sequencing involves randomly fragmenting DNA and sequencing the fragments. The benefit of shotgun sequencing is that it provides sequenced fragments across the entire DNA molecule (Figure 2.23 (a)). However, because there are a finite number of reads sequenced and they are spread across the entire molecule, the depth of coverage at any nucleotide tends to be low in comparison to other methods, such as reduced representation.

2.8.2 Reduced Representation

Reduced representation is the generic term given to any method which seeks to increase read depth by sequencing a reproducible subset of the genome space, minimizing the amount of sequencing required. Often, this is accomplished by fragmenting all or a portion of the DNA space using a restriction enzyme based digestion [28, 7]. Restriction enzymes are enzymes which cut the DNA strand based on a recognized pattern of nucleotides in a process known as digestion. For example, the restriction enzyme *Eco*RI recognizes the nucleotide pattern GAATTC and cuts the DNA between the G and the A nucleotides as in the example in Figure 2.22 [97].

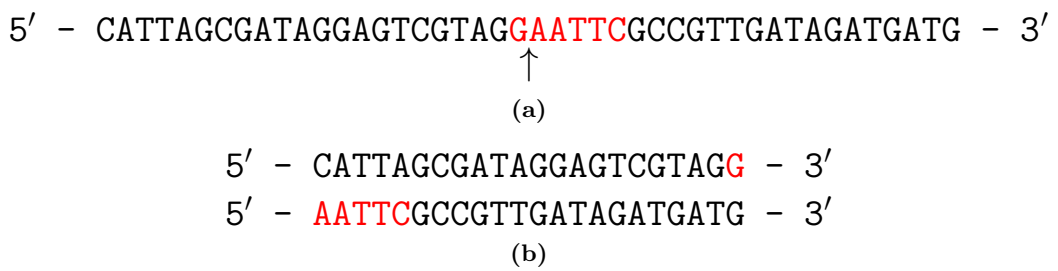

Figure 2.22: Digestion (a) of the DNA sequence CATTAGCGATAGGAGTCGTAGGCCGTTGATAGATGATG by *Eco*RI (restriction site highlighted in red and cut site shown with arrow) results in two fragments (b), CATTAGCGATAGGAGTCGTAGG and ATTCGCCGTTGATAGATGATG.

The benefit of reduced representation libraries are that they focus sequencing on specific areas of the genome resulting in greater depth with less overall cost. However, the pitfall of reducing overall genome coverage is that some variations with important functionality may be missed. Reduced representation libraries can be generated for any sequencing platform using a variety of methods [4, 28, 79]. Two specific methods, 3′ Transcript Profiling and Restriction-site associated DNA (RAD), will be discussed in greater detail next.

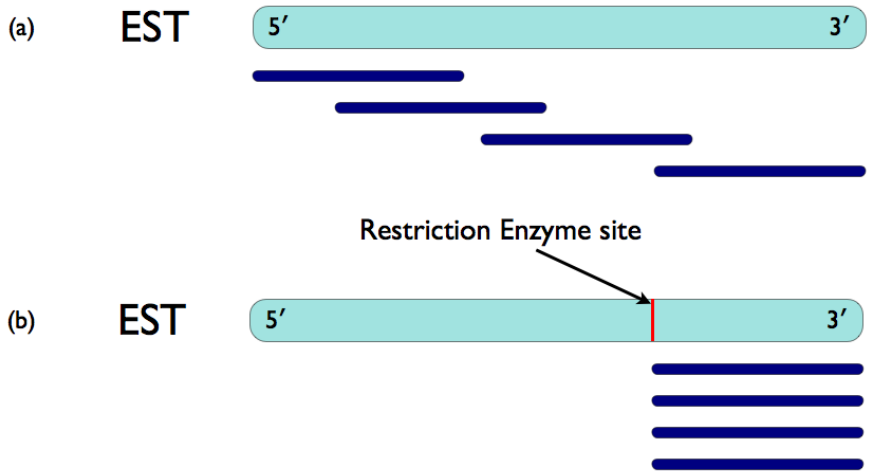

Figure 2.23: Comparison of sequencing results from (a) Shotgun sequencing (b) 3′ Transcript Profiling.

3′ Transcript Profiling

3′ transcript profiling is a restriction enzyme based method which captures the 3′-UTR of mRNAs. Capturing the 3′-UTR results in the sequencing reads being stacked at the 3′ end of the transcript sequence, resulting in greater depth for SNP discovery (Figure 2.23). Libraries are prepared by anchoring the 3′ end of the RNA sequence to a magnetic bead, digesting it with the restriction enzyme, washing away any unanchored fragments, and ligating on the sequencing primer [28]. The captured fragments are then ready for sequencing (Figure 2.24). By limiting sequencing to only the captured fragments the overall representation of the genome is reduced. However, the coverage of the regions sequenced has increased resulting in more robust SNP discovery.

Restriction site Associated DNA

Restriction site associated DNA (RAD) is another restriction enzyme based method. Unlike the 3′ Transcript Profiling method, the entire genome is digested and adapters containing a forward amplification primer, a sequencing primer, and a barcode (used to identify pooled samples) are attached to either side of the restriction site on the newly generated fragments. These fragments are then randomly sheared and a second adapter attached to the newly sheared end (Figure 2.25). Fragments with both adapters are amplified and then sequenced, reducing the areas of the genome that are sequenced while increasing the depth for more robust SNP discovery (Figure 2.26) [7].

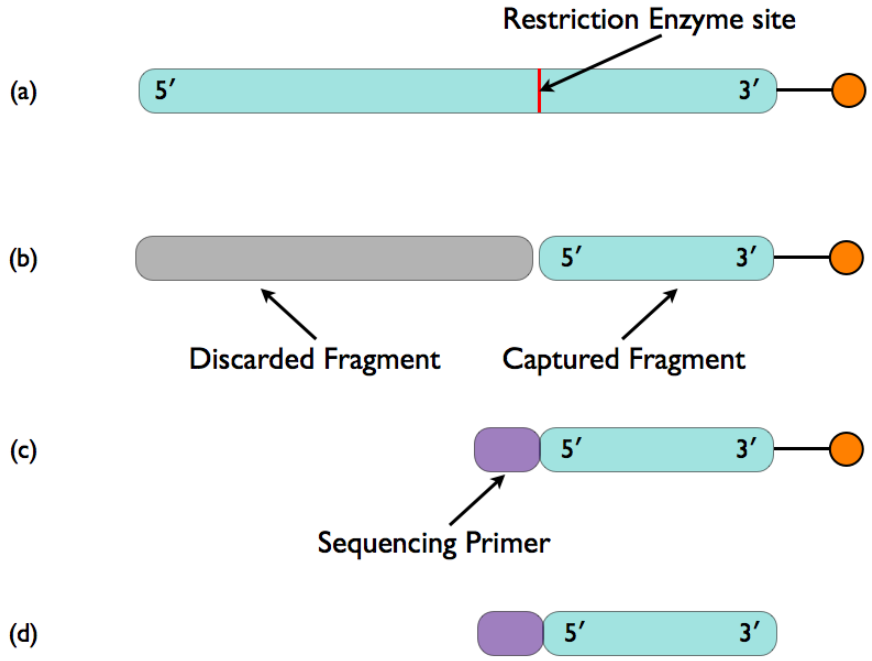

Figure 2.24: 3′ Transcript Profiling library preparation by (a) attaching the RNA molecule to a magnetic bead (b) fragmenting using a restriction enzyme (c) washing away any unanchored fragments and attaching the sequencing primer (d) eluting the fragment to be sequenced.

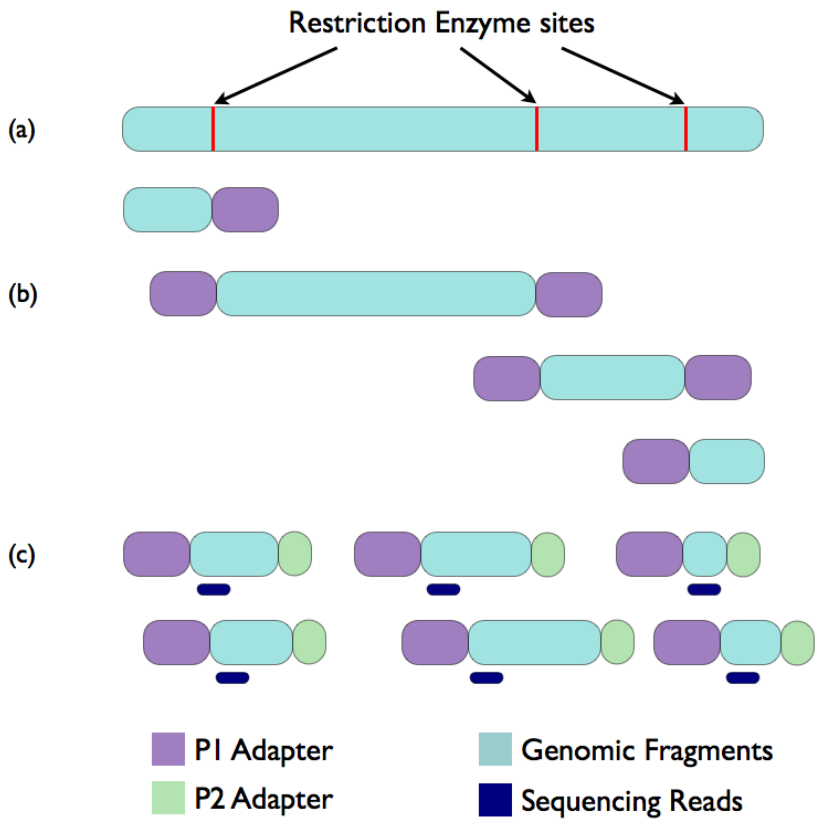

Figure 2.25: RAD library generated from (a) a genomic sequence fragmented using a restriction enzyme results in (b) a series of fragments with P1 adapters attached at the restriction sites. These fragments are then sheared into random sizes and a P2 adapter attached, resulting in sequencing reads adjacent to the restriction site (c).

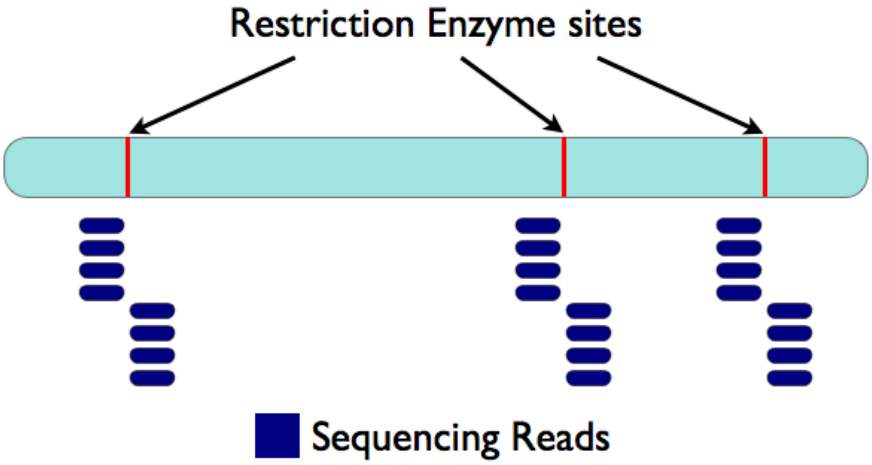

Figure 2.26: Sequenced reads are generated on either side of the restriction site resulting in a RAD tag with increased depth of coverage.

Chapter 3

Automated Intron-Spanning Primer Design For High-Throughput SNP Discovery Using Sanger Sequencing[1]

3.1 Introduction

Robust and automated PCR primer design is an important prerequisite to many strategies of large-scale discovery of nucleotide variation, specifically SNPs, as discussed in Section 2.6. At the time this thesis work was initiated, the primary method of SNP discovery in non-model organisms was to amplify a specific portion of a target genome using PCR primers. Often in more complex genomes a single gene from the related model organism can be represented by multiple members of a gene family. It is then desirable to co-amplify all members of the gene family using a single set of PCR primers. This requirement complicates the primer design, which tends to be further exacerbated by additional factors such as targeting non-coding regions of the gene sequence, which tend to be more polymorphic. This is especially complicated in organisms that do not have a fully sequenced genome, requiring additional time intensive experimental work to provide the necessary information for primer design. Thus, this phase of the SNP discovery method is often a bottleneck in the overall process.

Although next generation methods such as those described in Chapter 4 are now the predominant method for SNP discovery, the methods described in this chapter are still important for researchers performing experiments using targeted sequencing. One such application is the targeted sequencing of moderate numbers of genes where the cost of the methods outlined in Chapter 4 outweigh the cost of Sanger sequencing. The objective of the work in this chapter is to fully automate the currently used semi-automated pipeline (Section 3.2) for design of intron-spanning PCR primers in order to remove this process as a bottleneck to SNP discovery. Successfully completing this objective requires:

1. That there is not a significant drop in the efficacy of the PCR design process as measured by the PCR

[1] Much of the work in this chapter appears in [21].

primer amplification rate.

2. That the time to design primer pairs decreases dramatically, therefore removing the design phase as a bottleneck to SNP discovery.

Therefore, sub-objectives of this chapter are: to statistically validate the efficacy of the automated pipeline using experimental results from both the automated and semi-automated design pipelines and to measure the time required for the automated design of an arbitrary number of PCR primers in comparison to the semi-automated pipeline currently in use.

3.2 A Semi-Automated Pipeline For Design Of Intron-Spanning PCR Primers

Most frequently introns are targeted for SNP discovery as mutation rates in introns tend to be higher than in exons, as changes to the DNA sequence in introns do not affect the protein sequence of a gene. Therefore, targeting introns maximizes the number of SNPs discovered per sequenced base. Also, because mutation rates are lower in exons, there is less variation between exons of different individuals. Thus, allowing PCR primers designed for the exon sequence to amplify the same region in multiple individuals (Figure 3.1).

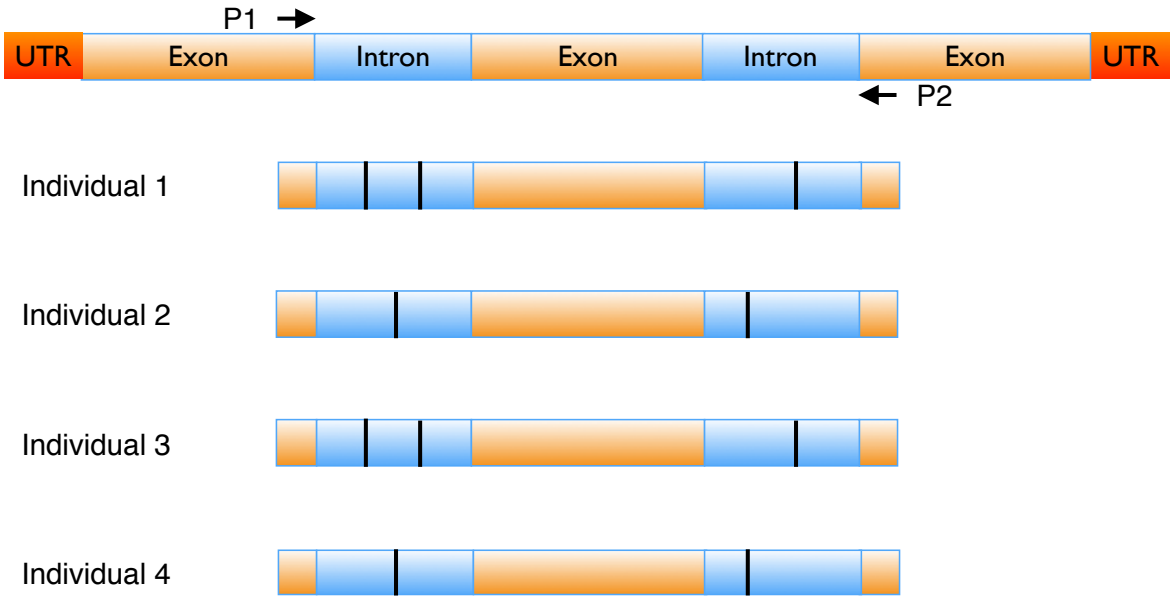

Figure 3.1: Amplification of a target region in multiple individuals using PCR primers (P1 and P2) designed in exons. SNPs are shown as black bars in the individuals where they are present.

Inferring the intron-exon structure of a non-model organism is usually most efficiently achieved by comparing EST sequences (with no structure) from the non-model to gene sequences, with carefully annotated structure, from the model species (Figure 3.2). This requires a number of steps, firstly, the EST sequences must be aligned to the model organisms gene using a method such as BLAST (Figure 3.3a). Next the aligned overlapping EST fragments are assembled into contiguous sequences (contigs) using a sequence assembler (Figure 3.3b). These contigs are then aligned back to the model sequence using a multiple sequence alignment program (Figure 3.3c). Gaps between contigs in the alignment to the model sequence represent introns, the size and position of these gaps can be combined with the positions of aligned contigs to infer the intron/exon structure of the non-model gene.

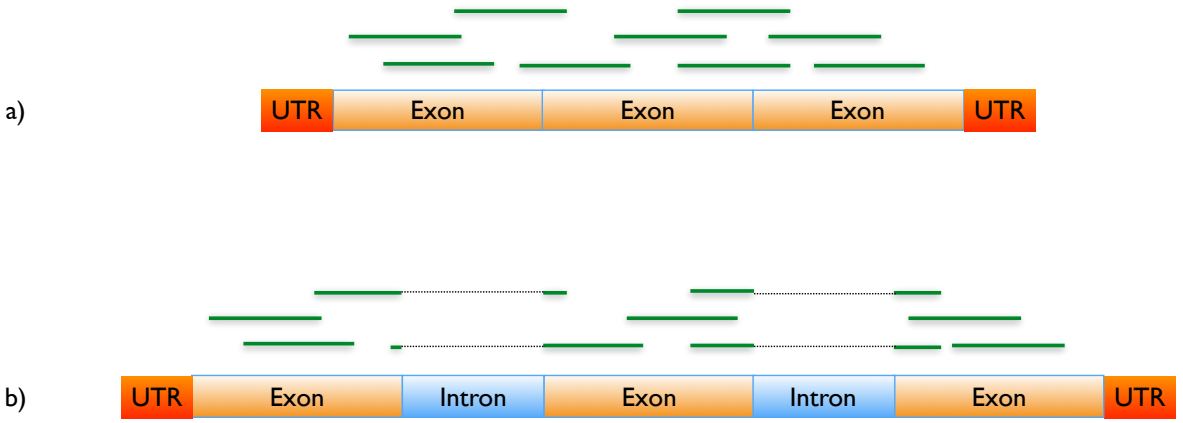

Figure 3.2: Alignment of the EST sequences to the intron-less cDNA sequence they were derived from (a) and aligned against the target gene from the model organism (b). Dashed lines represent gaps inserted into the EST sequences in sequences crossing intron-exon boundaries.

Previously, these steps were manually performed by a laboratory technician. The technician would perform the BLAST of the EST sequences versus the targeted model sequence. They would then extract the EST sequences from the FASTA file and import them into software (such as Gene Code's Sequencher) with a graphical interface where they could be assembled and aligned to the model sequence. The technician would then manually improve the alignment if necessary. Once the technician was satisfied with the alignment, they would scan the alignment for introns and manually choose sequence fragments for PCR primer design. This combination of technician input and software use (Figure 3.4), referred to as the semi-automated approach, is very effective but labour intensive.

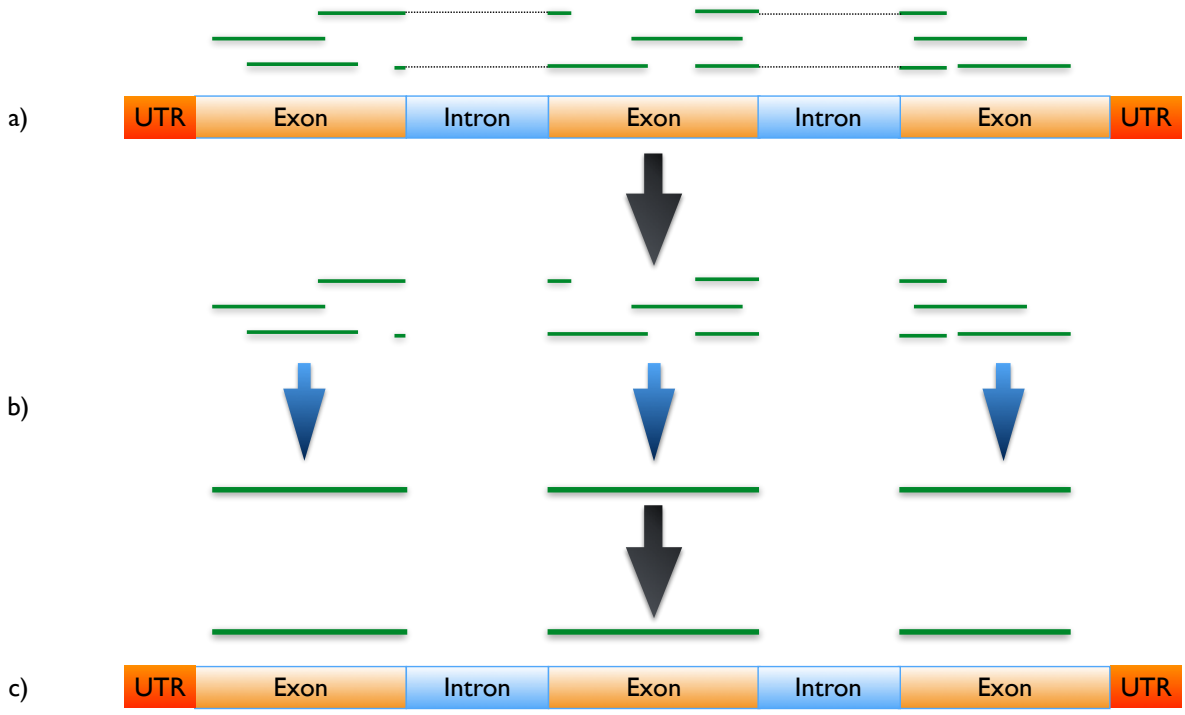

Figure 3.3: Inferring the structure of an unknown gene by alignment of EST sequences to a closely related model gene. EST sequences are aligned to the reference gene model (a) with solid lines representing the EST sequence and the dashed lines representing gaps inserted into the EST sequence due to introns. Overlapping EST sequence fragments are then assembled into contigs (b) and these contigs aligned to the reference model (c).

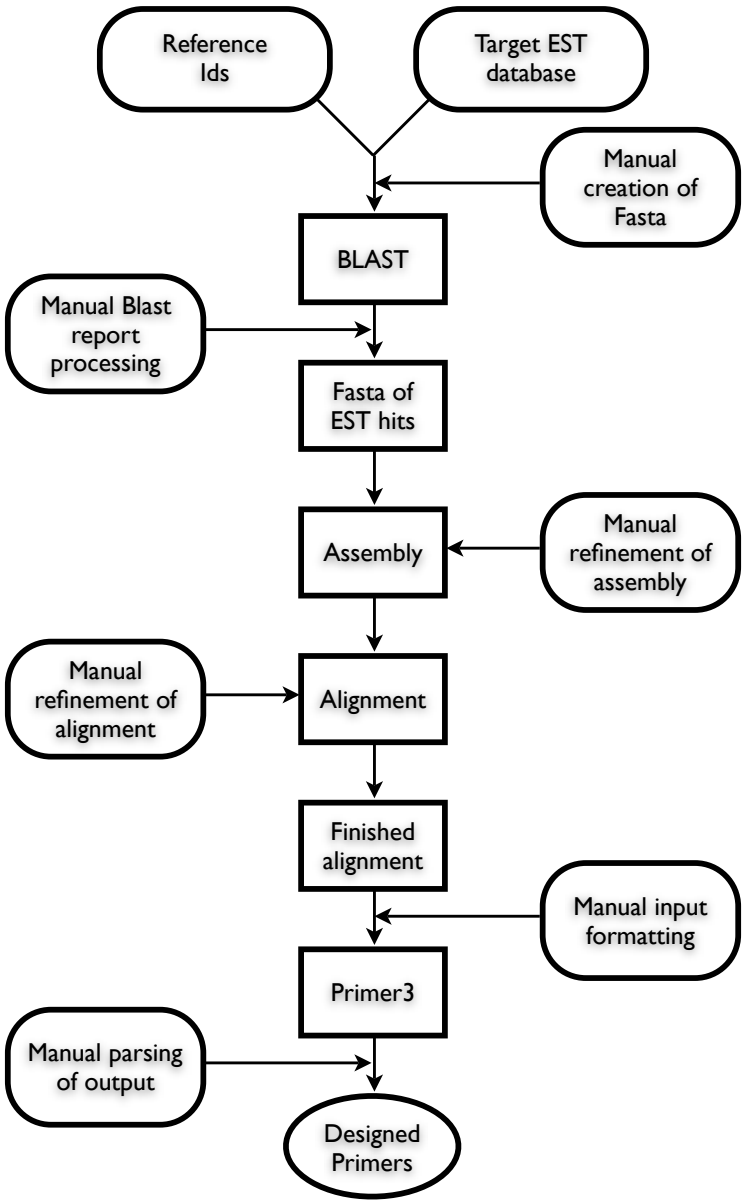

Figure 3.4: Flowchart outlining the steps and highlighting the requirement for user input of the semi-automated primer design approach.

3.3 Previous Work

Previous methods to automate primer design focused on automating the design of individual primer pairs for single targets as discussed by [123]. However, there were several programs available specifically for designing PCR primers for SNP discovery [123, 33, 32, 127, 39]. Each of these programs were not appropriate due to one of the following factors: the inability to infer the intron/exon structure [123, 33], the limited number of organisms that they can be used with [32, 39], or due to being unavailable [127]. Therefore, at the time, the only option for researchers was to perform the semi-automated primer design pipeline as describe, using a variety of bioinformatics tools to first infer the intron/exon structure and then design the primers. This often required manual refinement or parsing of the output of each phase of the intron/exon structure determination and primer design.

3.4 An Automated Pipeline For Design Of Intron-Spanning PCR Primers

The approach described here is to replicate the methodology of the semi-automated process performed by researchers in a fully automated non-interactive manner. This is accomplished by using available bioinformatics tools which perform computational steps that are similar to each stage in the semi-automated method (Figure 3.5).

For simplicity the pipeline requires only four input parameters: a list of reference sequence identifiers, the name of the EST BLAST database for the target organism, a cdbfasta formatted index of the EST FASTA file, and a cdbfasta formatted index of the reference sequences. To maintain flexibility the pipeline only requires that the reference sequence contain intronic and exonic sequence, which includes but is not limited to a sequenced bacterial artificial chromosome (BAC) or a gene model.

Output from each program is parsed and formatted for input into the next program using custom scripting in order to reduce the need for human participation. The pipeline uses the following open-source bioinformatics tools:

- **BLAST**
- **cdbfasta/cdbyank** - Cdbfasta is a program designed to create an index of a FASTA file in order to provide fast access to individual sequences and cdbyank is a program designed to utilize the index file created by cdbfasta to extract a sequence record from the FASTA file [89].
- **CAP3** - A *de novo* sequence assembly tool that can assemble long Sanger sequencing reads [41].

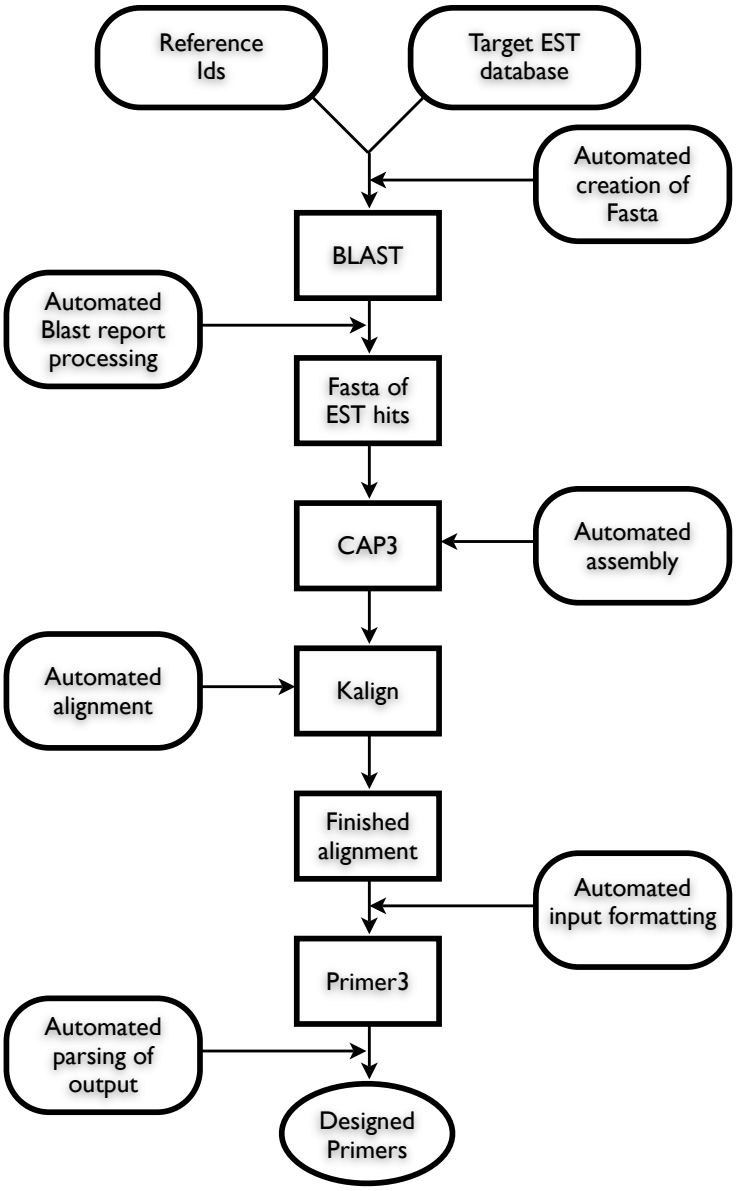

Figure 3.5: Flowchart of the automated primer design process showing the fully automated primer design approach with automated steps highlighted.

- **Kalign** - A multiple sequence alignment tool that allows for different terminal and non-terminal gap extension penalties [63].
- **Primer3** - A tool for designing PCR primers from a single DNA sequence [96].

Cdbyank is used in the pipeline to extract the desired reference sequences, the reference sequences are then used as input to BLAST. BLAST compares each reference sequence against the EST sequence database from the organism of interest. The BLAST reports are parsed to determine overlapping high-scoring pairs (HSPs), which represent the conserved sequence fragments between the EST sequences and the reference sequence. The conserved fragment of each EST sequence matching a reference sequence is then extracted from the input EST FASTA file using cdbyank and Perl's substring command. EST fragments from overlapping HSPs are output to a file and assembled into a conserved region using CAP3. These conserved regions (exons) are then aligned to the reference sequence using Kalign. From this alignment a consensus sequence can be produced with exons separated by inferred intronic regions (gaps in sequence coverage). Each nucleotide of the intron sequence is translated to the ambiguity character (N) (Figure 3.6). This allows Primer3 to design primers that span intronic regions. The consensus sequence is formatted into the Primer3 input format and Primer3 is run on each input file. The resulting output files are parsed to extract forward and reverse primer sequences and primer properties such as the predicted size of the amplified sequence.

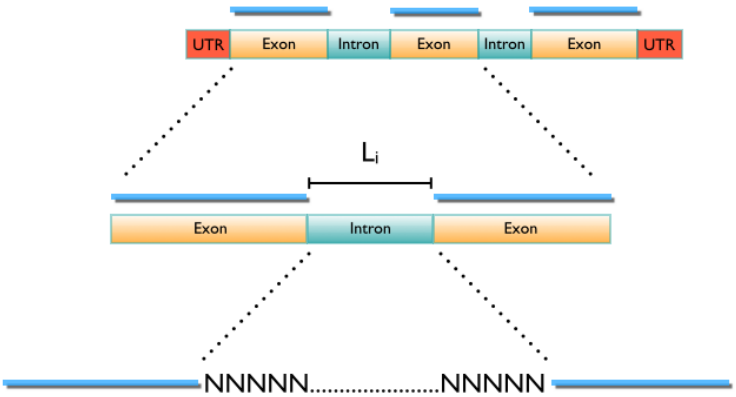

Figure 3.6: Illustration of how aligned reads can be used to infer the length (L_i) of an intron and the conversion of intron gaps to ambiguous Ns.

Optimization of the parameters for the major computational elements was done using prior biological knowledge. The parameters for BLAST include the BLAST program blastn, for nucleotide queries aligned to a nucleotide database, the FASTA of query sequences, the BLAST formatted reference database, and an E-value cutoff of 1e-5. The e-value cutoff is used to include all matches of reasonable significance, with some of these sequences possibly being removed in the subsequent assembly and alignment steps. Parsing of the BLAST output is performed using BIOPERL modules with HSPs below 85% identity being discarded.

CAP3 is run using a percent identity cutoff of 85% for co-assembled HSP sequences to generate the contigs for alignment to the reference. Although parameters were not altered systematically, after several iterations of parameter adjustment and checking of alignment results, the parameters for the multiple sequence alignment program Kalign were set as follows: gap open penalty equal to 80, gap extension penalty equal to 3, terminal gap extension equal to 0.5. Using a very low terminal gap extension penalty allows the alignment algorithm to more accurately place the full length of the contig within the reference sequence, which is desirable since the positions of the contigs are what is used to infer the intron structure. Primer design parameters had previously been optimized in the semi-automated approach and were therefore carried over to the primer design process.

3.4.1 Output Types

Two separate Perl scripts were implemented using the automated pipeline, one which output a single pair of primers for each reference sequence and a second which outputs multiple non-overlapping primer pairs for each reference. Primer3 outputs several primer pairs, separated into records by the string "||", ordering them based on how closely they match the desired input parameters. Therefore, in order to produce a single primer pair per input reference sequence, just the first record in the Primer3 output is processed (Figure 3.7 (a)). To produce multiple primer pairs, the Primer3 output is processed to remove overlapping primer pairs, resulting in a set of primers covering the target sequence (Figure 3.7 (b)).

3.5 Results and Evaluation

Results of PCR primer design using the automated approach are compared to results gathered for the semi-automated approach in [100], to compare the total design time and PCR amplification rate. PCR amplification rate is the proportion of primer pairs that amplify a sequence in the target species and provides an estimation of how well the intron/exon structure has been inferred. The hardware used for evaluation of the semi-automated and automated approaches is a HP Proliant DL-385 server running the CentOS 4.5 linux distribution. This machine houses two AMD 2.2 GHz processors and 16 GB RAM.

PCR primer design was done as part of a project to develop SNP markers for targeted genes in the crop species *Brassica napus* by amplifying the same DNA sequence from multiple individuals, sequencing the resultant fragments, and comparing the results to find SNPs. Genes from the model organism *Arabidopsis thaliana*, which shares a level of sequence similarity of 87% in exons [16] with *Brassica napus*, were used in evaluating the two approaches.

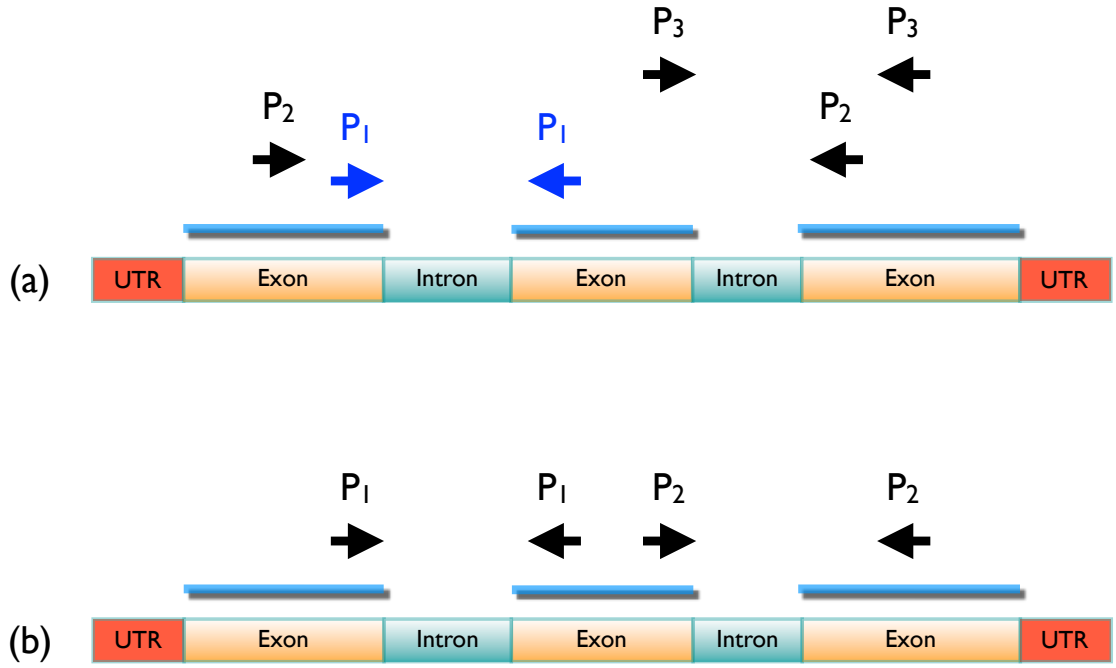

Figure 3.7: Primers can be designed to generate a single amplified fragment per contig (a), usually the most optimal of a set of primers (P_1) is chosen. Primers can also be designed to amplify across the gene space (b).

3.5.1 Semi-Automated Pipeline PCR Results

For the semi-automated approach 133 target genes were selected. Primer pairs were designed for all 133 target genes and PCR carried out resulting in 117 (88%) successful amplifications [100].

3.5.2 Automated Pipeline PCR Results

Single Primer Pair

For the automated single primer pair approach, 256 *Arabidopsis* gene models were selected for input to the pipeline. Of the 256 selected gene models, 253 (98.9%) successfully had primers designed for them. Further, PCR was performed using the resulting primer pairs, with successful amplification occurring in 206 primer pairs (81.5%). The 206 successful amplifications represent an overall success rate of 80.5% for all gene models submitted to the pipeline.

Multiple Primer Pairs

Multiple primer pairs were designed to span 22 target sequences. There were 90 primer pairs chosen in a amplification product size range of 200-1100 base pairs in order to maximize the coverage of the target

sequences. Of these 90 primer pairs there were 83 successful PCR amplifications resulting in 92% efficacy.

3.5.3 Evaluation

In order to compare the efficacy of the semi-automated and automated pipelines the number of successful PCR amplifications was compared to the number of non-successful amplifications using a 2x2 contingency table and the Pearson's Chi Squared (χ^2) statistical test. For the automated method, the results of both the single primer pair and multiple primer pair methods were combined. The null hypothesis for the chi squared test was that the two primer design pipelines have the same efficacy. The chi squared test was performed using the R programming language's built in function and resulted in a $\chi^2 = 1.0536$ and a p-value = 0.3047. Based on this p-value, the null hypothesis cannot be rejected, therefore the efficacy of the two pipelines does not differ.

Time requirements to design 100 primer pairs using the semi-automated approach were compared to the time required to design 100 primer pairs using the fully automated pipeline. In our tests the automated pipeline designed 100 primers in less than 10 minutes (an average of 9 minutes and 16 seconds over 5 trials). This is a significant improvement over the semi-automated approach, which required approximately 1 hour to produce 2 primer pairs or 50 hours per 100 primer pairs [100].

Additionally, the sizes of the amplified products for all of the methods were checked using gel electrophoresis and the majority were within the size range predicted by the alignment of the conserved fragments to the model gene sequence. Variations in the sizes of the products versus the expected sizes are explained by variation in the intron sizes between the model organism and the target organism.

3.6 Discussion

As evident by the results of the evaluation, the pipeline greatly reduces both required user input and primer design time. Further, the results of the statistical analysis show that there is no significant decrease in the efficacy of the automated method when compared to the semi-automated method. Therefore, the work provided in this chapter successfully removes PCR primer design as a bottleneck to SNP discovery. This software has been demonstrated to work in the crop *Brassica napus* but should be equally effective in any non-model organism where an EST sequence resource is available or could be generated, and where a closely related model organism is present.

Further, a very similar method was published in 2009 by You *et al.* in parallel to this work [127]. Their method is very similar to the automated pipeline described in this chapter and has been proven to work in wheat, indicating that the method works across multiple non-model organisms. The method provided by You *et al.* differs from what is described in this chapter in two main ways. First, it removes primer pairs

in the output that will amplify more than one gene from a gene family by aligning the primer sequences to a set of non-redundant EST sequences using BLAST and removing primers that match to more than one EST sequence. This is a difference in the methodology employed for SNP discovery and does not indicate a problem with either method. Second, all potential primer pairs are output and the user then does a manual selection of a particular primer pair. While this approach does allow for single or multiple primer pairs to be selected, overlapping primer pairs are not removed, which would make the selection of multiple primer pairs more tedious.

Chapter 4

Methods For The Generation Of SNP Genotyping Arrays In non-Model Diploid and Polyploid Species[1]

4.1 Introduction

The development of next generation DNA sequencers has led to a vast wealth of DNA sequence data and has significantly reduced the cost of generating new sequence data. It is therefore more important than ever to develop methods for the analysis of this data. Due to the volume of data being produced, these methods can no longer rely on researchers to manually process the data. For this reason, computational methods for the analysis of next generation sequencing data are crucial to many researchers in the biological sciences.

Mining DNA sequence data for genetic variation, such as SNPs, is an important tool for researchers in fields such as human health and plant breeding. Development of SNP resources typically requires a set of high quality reference sequences. In the past, the lack of such high quality reference sequences was a limiting factor for SNP discovery in non-model organisms. However, recent advances in DNA sequencing and *de novo* sequence assembly has made it possible to generate a set of high quality reference sequences for many non-model organisms [51, 70, 87, 121].

Many computational methods have been developed for the discovery of SNPs in next generation sequencing data (refer to Section 2.6.3). These methods involve taking sequenced reads from an individual and mapping them to a set of DNA reference sequences. The resulting alignment can be scanned for variants. Often statistical methods are employed, using factors such as read depth and sequence quality, to generate a probability of a called variant being real. False positives are mainly caused by sequencing errors and misalignment of reads to the reference sequence [78, 9, 8]. Even with these advanced SNP discovery methods, the effective application of these SNPs to research projects often requires additional computational filtering and selection steps, particularly in organisms with complex genomes [120, 17, 119]. The application that this work focuses on is the development of SNP genotyping arrays.

For SNP genotyping arrays there is typically a minimum number of chips that must be ordered resulting

[1] A subset of the work from this chapter has been published in PLoS ONE [19] and BMC Genomics [99].

in significant cost and changes to the array cannot be made easily. Therefore, it is particularly important to post-process discovered SNPs in order to ensure the most robust genotyping array possible. The first stage of developing a SNP genotyping array is to generate SNP data from multiple individuals. SNP calls can then be combined from all of the individuals into a comprehensive variant list. By combining evidence from multiple individuals, a more robust screening of available variants can be accomplished.

The goal of the work described in this chapter is to provide methods for utilizing large amounts of next generation sequencing data for the development of SNP genotyping arrays in non-model diploid and polypoid species. As such, a general workflow for the design of SNP genotyping arrays in non-model organisms will be presented. Based on this array design workflow, available tools (and their limitations) can be discussed. The methods described in this chapter seek to address the limitations of available software for the development of high quality SNP genotyping arrays.

First, the development of three algorithms for combining SNP data from multiple individuals (all addressing step (b) of Figure 4.1) will be discussed along with evaluations of their computational time and space complexities and experimental performance. Evaluation of each algorithm's time and space complexities provides an estimation of the efficiency of the algorithms, while the evaluation of the experimental performance provides confirmation of the complexity analysis, without ignoring constant factors so important to these typically large data sets. Further, evaluation of the experimental performance allows for statistical estimates of resource requirements for future experiments. As two of the three algorithms are designed to utilize parallel CPU threads to process a portion of their input, potential bottlenecks of the parallelization process are discussed. Using the analysis of the algorithms, recommendations for algorithm selection are provided based on input sizes and computing resource capacities.

Next, implementations for SNP filtering (step (c) of Figure 4.1) and selection (step (d) of Figure 4.1) are discussed as important post-processing steps of SNP discovery and as important pre-design steps for design of SNP genotyping arrays. In particular, two methods for the selection of SNPs for inclusion onto a SNP genotyping array are discussed. Then, results from SNP genotyping arrays using the methods from this chapter are discussed and evaluated for statistical significance, in terms of number of polymorphic loci (genomic positions) and number of clean genotyping clusters, based on comparisons to other SNP genotyping arrays. Finally, conclusions about the overall effectiveness and utility of the methods from the chapter are discussed.

4.2 SNP Array Design

A general workflow for the creation of a SNP genotyping array starting with the sequenced reads from multiple individuals and a set of reference sequences is presented in Figure 4.1. The process begins with the independent mapping of sequencing reads from each individual against the set of reference sequences.

SNP detection can then be performed for each individual, resulting in a SNP report and SAM alignment file (a common file format for read mapping data, see Figure 4.3 (b)) for each individual (Figure 4.1 (a)). SNP and alignment data can then be combined into a single output file (SNP summary table, Figure 4.2), where each line of the file represents a SNP called in at least one of the individuals (Figure 4.1 (b)). The development of the SNP summary table serves two purposes: it provides the first aggregate view of the data to the researcher, and provides a type of checkpoint after the most computationally intensive process in the workflow. The summarized SNP data is then filtered, using evidence from multiple individuals, for SNPs that represent potentially robust markers (Figure 4.1 (c)). Finally, a set of SNPs can be chosen for inclusion onto the array from the filtered SNPs (Figure 4.1 (d)). Selected SNPs are then submitted for array design by providing the SNP flanking sequence to the array manufacturer. A SNP's flanking sequence consists of two substrings of the DNA sequence (one occurring immediately to the left of the SNP and the other immediately to the right of the SNP) joined by the reference/SNP pair delimited by braces (Figure 4.4(b)). The SNP flanking sequence is used to design the probe sequence (Section 2.3.1) that will be included on the array; the amount of flanking sequence required is determined by the array technology being used.

This work provides new algorithms for only a subset of these steps; that is, the aggregation of SNP data from multiple individuals into a single output file, the filtering of SNP data, and the selection of SNPs for inclusion onto a SNP genotyping array.

4.3 Available Tools

The read mapping process (step (a) of Figure 4.1) can be accomplished using one or more of the programs mentioned in Section 2.5.4. However, currently the most commonly used programs are Bowtie2 [61] and SOAP [71]. The three most widely used software packages for SNP calling in multiple individuals are samtools (mpileup) [67], the genome analysis toolkit (GATK) [82], and SOAPsnp [71].

SOAPsnp, originally made available in late 2008 is not a true SNP discovery software package. SOAPsnp's purpose is to generate a consensus sequence based on alignment of reads to a known reference sequence. SNPs can then be discovered by comparing the consensus sequence to the reference sequence. There are several limitations to using SOAPsnp: it is limited to using only Illumina sequence data, only supports one specific non-standard alignment format called the SOAPaligner alignment format, and does not provide a simple method for determining the depth of reads resulting in a SNP call [71].

To call SNPs in multiple individuals using samtools, the mpileup tool is called on BAM files (the binary version of SAM files). This method was not available when we began our work, as multisample SNP calling was not available in samtools until October 28, 2010 when mpileup was first included in samtools release 0.1.9 [68]. Additionally, multisample SNP calling in mpileup is limited to samples that are biallelic, meaning that only two different nucleotides are expected across all samples. This is true in certain instances, such as

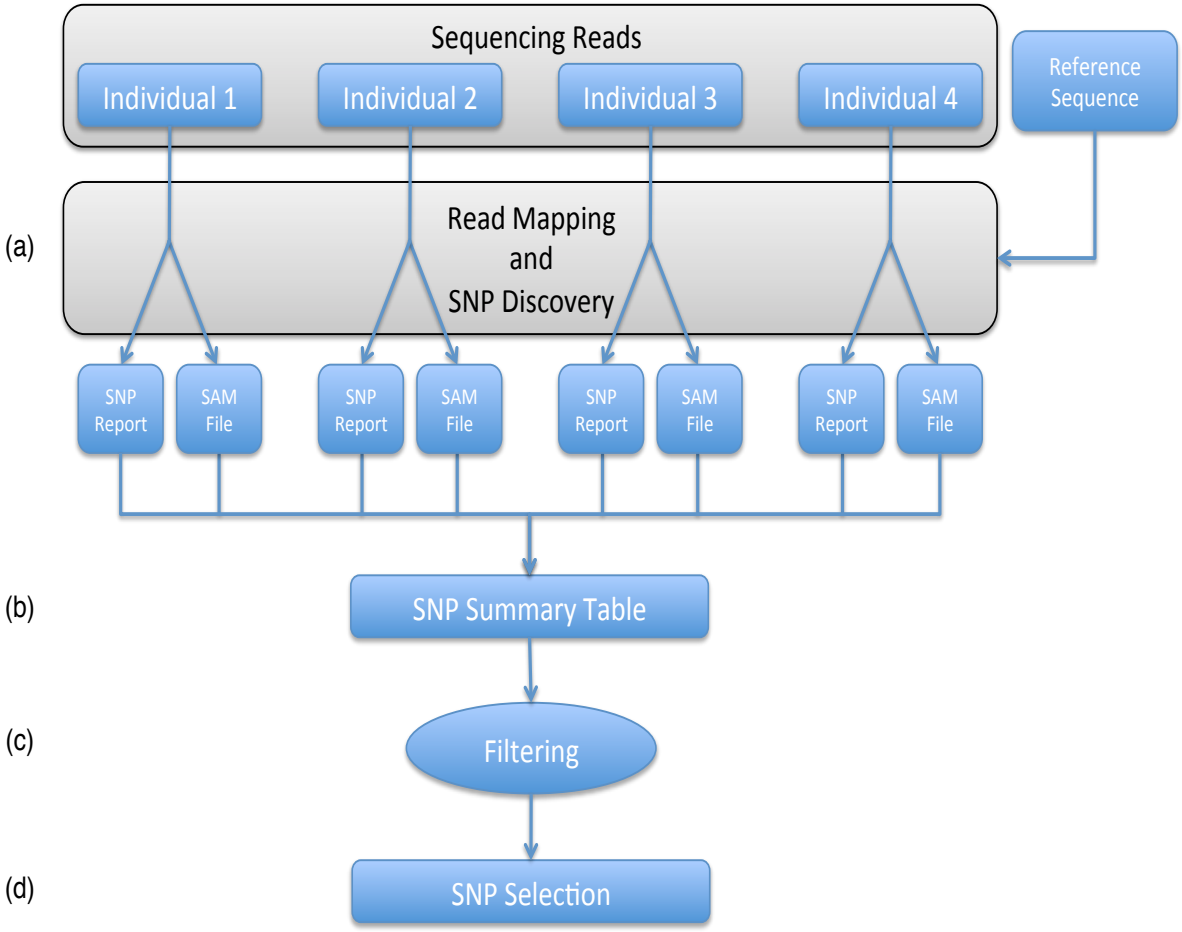

Figure 4.1: Workflow for the design of a SNP genotyping array. The process begins by independently mapping sequencing reads from multiple individuals to the reference sequence set and calling SNPs in the alignment (a). SNP reports and SAM files are then combined to generate a SNP summary table (b). This table is filtered to remove SNPs that will not result in robust markers (c). Finally, a group of SNPs can be selected from the filtered set for inclusion onto the array (d).

SNP id	Ref Seq	Ref pos	Flanking Seq	Ref base	Ind.1 Call	Ind.1 Freq	Ind.1 Depth	Ind.2 Call	Ind.2 Freq	Ind.2 Depth	Ind.3 Call	Ind.3 Freq	Ind.3 Depth
Ch1_100	Chr 1	100	AT...GA[C/A]AA...TA	C	A	100	20	C	100	12	X	X	X

Figure 4.2: Sample output illustrating the difference between variant, reference, and null calls (individuals 1, 2 & 3 respectively). In this example, individual 1 has a variant called at position 100 in chromosome 1, while individuals 2 and 3 have no variation detected. By querying the alignment of reads to the reference, it is determined that individual 2 has 12 reads aligned at position 100 of chromosome 1 and is therefore the same as the reference (C) and that individual 3 has no reads aligned to the reference and is therefore a null (X) call.

in mapping populations, where alleles are inherited from one or both parents, but is not true for the diverse samples used to generate SNP genotyping arrays. Further, when calling SNPs with mpileup there is limited information reported for each individual; only a total depth of reads and a probability that the individual is either the same as the reference (homozygous for the reference), variant (homozygous for the variant), or has both the reference and variant alleles (heterozygous). However, in complex genomes it is useful to have more detailed information, such as the number of reads supporting a reference or variant call, in order to more accurately review SNP calls.

In comparison, the GATK SNP calling method which has a tool called DepthPerAlleleBySample, provides sufficient biological information for use in complex genomes by reporting the read depth of both the reference and alternate bases (alleles) for each individual. However the DepthPerAlleleBySample tool was not released until July 2012 and was therefore not available when the thesis work was initiated [80].

An important factor missing from all of these algorithms is the ability to produce flanking sequence information for each SNP. Since the flanking sequence is used to generate the probe sequence which represents the SNP on the array, generation of flanking sequence for each SNP is a vital step in designing a SNP genotyping array. Additionally, the ability to post-process called SNPs is very limited. There are no available software packages for standalone filtering of combined SNP output from SOAPsnp or mpileup. However, GATK does provide a limited method for filtering combined SNP results [81]. Further, no software is currently available which assists in the selection of SNPs for a genotyping array.

4.4 Building A Comprehensive List Of SNP Calls Across Multiple Individuals

As the generation of read mapping and SNP calling algorithms is beyond the scope of this thesis, the approach taken for the generation of alignment and SNP data is to align next generation DNA sequencing reads to high quality reference sequences using the proprietary CLC Genomics Workbench's *map reads to reference* tool

(CLC Bio Inc, Aarhus, Denmark). These alignments were performed using mapping parameters dependent on the species being studied, with parameters chosen based on prior knowledge of the species and its affects on alignment [86]. Sequencing reads from each individual were independently aligned to the reference and the alignment exported in the SAM alignment format. From each of the independent alignments, SNPs were called using the CLC Genomic Workbench's *probabilistic variant detection* tool. The resulting variant calls were then exported as tab delimited text files.

In order to allow the read mapping and SNP discovery phases of the workflow to be accomplished using a variety of tools, the input formats chosen for the algorithms described in Sections 4.5 and 4.6 are either current industry standards (SAM alignment format) or easily generated (tab-delimited text format). Thus, these algorithms will work with any mapping software that can output alignment data in the SAM format and any variant calling software that produces text output. During the course of the thesis work, the variant call format (VCF) for SNP reports became an industry standard. The ability to transition from the tab-delimited text format to the VCF format will be discussed in Chapter 5.

The algorithms detailed in Sections 4.5 and 4.6 are pipelines which address the generation of the SNP summary table (step (b) within the higher level pipeline of Figure 4.1). The algorithms generate identical output, however the two algorithms in Section 4.6 differ in parallelization from the algorithm in Section 4.5. All of the algorithms differentiate reference (aligned reads indicate the same base as the reference) and null (no aligned reads at the reference position) calls for reference positions in individuals where no variant has been called, but evidence of variation exists in other individuals, using alignment data from SAM alignment files. The resulting output is a table containing a unique SNP id, the reference sequence where the SNP was found, the SNP's position in the reference, available flanking sequence information, the reference base, and SNP call data for each individual (Figure 4.2). The unique identifier and SNP flanking sequence are used by the array manufacturer, the SNP call data can be used for the filtering of SNPs and the reference sequence and SNP position can be used for SNP selection purposes.

4.5 Serial SAM Processing Algorithm

When the thesis work was initiated, next generation sequencing was in its infancy. While the cost per base pair sequenced provided by early next generation sequencers was much less than that of Sanger sequencing, the total cost of sequencing was still a limiting factor in the number of individuals for SNP discovery. The initial algorithm developed in this thesis for processing alignment data (to be described in Algorithms 4.5.1 - 4.5.6) places more emphasis on the identification of robust SNP markers rather than performance; however, it is still quite practical on modern computer hardware when the number of individuals is low (less than 10). We have used the term *serial SAM processing* for this algorithm, as it processes each SAM file one after the other, as opposed to the parallel SAM processing algorithms described in Section 4.6.

The main method of the algorithm is described at a high level in Algorithm 4.5.1 and has four required input parameters: a directory that contains one tab delimited SNP report per individual, a FASTA[2] format file containing all of the reference sequences used in the alignment process, a directory containing the SAM format alignment files, and the name of a file to which results can be written. As the SAM and SNP report files for an individual must be linked together, the individual's identifier is required by the algorithm to be present at the beginning of the file names.

The pipeline (to just complete step (b) of the higher level pipeline described in Figure 4.1) contains several main components (a section expanding upon each step, along with the formal pseudocode is provided below). The first step is to parse and collect the SNP information from the SNP reports (Algorithm 4.5.1 (i)), next the alignment files (SAM files) are processed to retrieve reference calls if available (Algorithm 4.5.1 (ii)). As this algorithm is developed to generate a SNP list for selecting SNPs for a genotyping array, flanking sequence is required. In order to retrieve the flanking sequence information the reference sequences are first parsed from the FASTA file given on the command line (Algorithm 4.5.1 (iii)). Reference sequences are stored in a hash using the sequence name as the key and the sequence as the value. The subroutine ADD_MAJOR_VARIANT (Algorithm 4.5.1 (iv)) then traverses the SNP calls and adds a field that contains the variant present in the majority of individuals. This field is then used when generating the flanking sequences (Algorithm 4.5.1 (v)) as described in Algorithm 4.5.5. Finally, the SNP and flanking sequence data is passed to the OUTPUT_TABLE subroutine (Algorithm 4.5.1 (vi)) where it is formatted and printed as described in Algorithm 4.5.6.

4.5.1 Processing SNP Reports

The first processing step is to parse all of the SNP reports to find all reference positions with a variant called in at least one individual. Algorithm 4.5.2 describes the subroutine PROCESS_SNP_REPORTS, which takes a list of reports found in the SNP_dir input parameter. The algorithm loops over the files, extracting the name of the individual from the file name. It stores the names of the files in an array which is used for ordering the results in Algorithm 4.5.6. For each line of the tab delimited file, the line is split into various fields and as long as the variant type is "SNP" the information is stored in a hash. Other variant types, such as insertions/deletions, are ignored as they are not desired for this application. Since all individuals are aligned to a common reference, SNP positions are stored using the name of the reference sequence (ref_id) as the first key of the hash and the position in the reference (pos) as the second key in the hash. The individual's identifier (ind_name) is then used as a third key to separate results from different individuals. The hash, which now contains all of the SNP positions for all of the reference sequences across all of the

[2]a common file format for sequence data containing a description line (starting with > followed by the sequence identifier) followed by one or more lines of sequence data

individuals, is then returned.

4.5.2 Processing SAM Alignment Files

The next step is to process all of the alignment data so that individuals without a variant call at a particular position can be called as either the same as the reference or null (no reads aligned). Algorithm 4.5.3 describes the subroutine PROCESS_SAM_FILE which is called in the main method on each SAM file found in the SAM_dir input parameter. The algorithm opens the file and for each line that contains alignment data (there are several header lines that need to be skipped) processes the alignment data if the read is aligned to a reference sequence where a SNP has been identified. Skipping reads aligned to SNP-free reference sequences reduces running time by avoiding processing uninformative alignments.

Each read alignment contains four important pieces of information: the reference sequence id (ref_id), the start position of the alignment in that reference ($start\_pos$), the compact idiosyncratic gapped alignment report (CIGAR) string ($cigar\_string$), and the read sequence ($read\_seq$) (Figure 4.3(b)). The $cigar\_string$ (Figure 4.3(b), highlighted in red) is a sequence of operations and their counts describing how subsequences of the read align to the reference. In order to get the actual alignment of the bases of $read\_seq$ to the reference, the operations of the $cigar\_string$ must be processed as described in Algorithm 4.5.4. For this algorithm, five operation types are important as they change the alignment of bases in the read sequence to the reference. These operations indicate:

1. The subsequence of the read is matched to the reference (operations M,X,=). M represents an alignment match that is either a sequence match or mismatch, while = is a sequence match and X is a sequence mismatch.

2. Deletion of a subsequence in the read versus reference (operation D). Equivalent to adding a gap character to the read sequence at the aligned position(s).

3. Insertion of a subsequence in read versus reference (operation I). Equivalent to adding a gap character to the reference sequence at the aligned position(s).

4. Soft clipping of a subsequence of the read (operation S). Soft clipped bases are bases that are in the read sequence ($read\_seq$) presented in the SAM file but are not used in the alignment of the read to the reference. This is opposed to hard clipped bases (operation H) which are removed from the read sequence ($read\_seq$) as presented in the SAM file. Clipped bases are often low quality and therefore not used in downstream analysis. The determination of low quality and the decision to perform soft or hard clipping is done by the alignment algorithm.

5. Skipped reference bases (operation N). The N operation is used to represent introns in alignments of mRNA sequences to genome sequences and is not defined otherwise.

For a full list of CIGAR operations, refer to the SAM specification [113]. Figure 4.3(a) gives an example alignment of a read (r001) to the reference sequence (Ref) with the resulting SAM file shown in Figure 4.3(b). The CIGAR string for this alignment is 2S4M2D2M1I1M2N3M and indicates that the read starts with 2 soft clipped bases, followed by 4 aligned bases, a 2 base pair deletion, 2 aligned bases, a single base insertion, a single aligned base, 2 skipped reference bases, and ends with 3 aligned bases.

a)

```
Coords          0000000001111 111111222
                1234567890123 456789012
   Ref          ATTGCGACCTGTT*GCAAGCTAG
  r001             gcGACG**TTAG..AGC
```

b)

```
@SQ SN:Ref LN:22
r001 0 Ref 6 30 2S4M2D2M1I1M2N3M 0 0 GCGACGTTAGAGC *
```

Figure 4.3: An example alignment of a read (`r001`) to the reference (`Ref`) (a) and the resulting SAM alignment (b). The SAM file begins with header line(s), which begin with the @ symbol. In the example, a single header line giving the name and length of the reference sequence is provided. Following the header line there is a line for the alignment of each read. These lines are tab delimited and have the following important fields for our algorithm: reference name (field 3), start position (field 4), CIGAR string (field 6 and highlighted in red), and read sequence (field 9).

4.5.3 CIGAR Alignment Processing

As the CIGAR string present in the SAM alignment is only a representation of an alignment of a read to the reference, which does not contain the per base detail required, the CIGAR string must be decoded to determined the per base alignment of the read to the reference. Algorithm 4.5.4 describes in detail the process of parsing the CIGAR string to determine alignment of a read to the reference. It begins by setting the reference position ref_pos to the start position ($start\_pos$) of the read alignment, the position in the read sequence (seq_pos) to zero, and then looping over the CIGAR operations in *ops* (for our example in Figure 4.3, *ops* would contain S,M,D,M,I,M,N,M). The actions taken in processing of the alignment data are dependent on the operation observed.

If the current operation is M, X, or = then the number of times (i) the operation needs to be performed is retrieved from *op_counts* (e.g. 2,4,2,2,1,1,2,3). The algorithm iterates from one to j, pulling out the base call in the read from *seq* (an array where each element is a single base from the read sequence *read_seq*) using *seq_pos* as an index. If a SNP has been observed in another individual at ref_pos, then the total read depth at ref_pos (Algorithm 4.5.4 (i)) and the base call count (count of this particular allele at ref_pos) (Algorithm 4.5.4 (ii)) are incremented (indicating coverage). If a SNP has not been observed at ref_pos in

any other individual, no alignment information is recorded (since this base position is uninformative for SNP discovery purposes). The algorithm then increments the position in the reference (ref_pos) and the read (seq_pos) before performing the next iteration of the operation. If no further iterations of the operation are available the next operation is read from *ops* and the process starts over.

If the current operation is D then the number of times (j) the operation needs to be performed is retrieved from *op_counts* (e.g. 2,4,2,2,1,1,2,3). The base call is set to the gap character (*) for each iteration. The algorithm iterates from one to j, if a SNP has been observed in another individual at ref_pos, then the total read depth (Algorithm 4.5.4 (iii)) and base call count (Algorithm 4.5.4 (iv)) are incremented as above. Unlike operations M, X, and =, when the operation is D only ref_pos is incremented at the end of each iteration as the position in the read has not changed.

If the operation is S or I, only the position in the read sequence requires updating as the reference sequence position has not changed. Therefore, the algorithm adds the number of operations found in *op_counts* to *seq_pos*. If the operation is N, the algorithm adds the number of operations found in *op_counts* to ref_pos as only the position in the reference sequence requires updating since the position in the read has not changed.

4.5.4 Generating Flanking Sequences

Flanking sequences are substrings of the reference sequence to the left ($5'$) and right ($3'$) of the SNP position (Figure 4.4(a)). The process of generating flanking sequence information for each SNP is detailed in Algorithm 4.5.5. For each SNP position the reference sequence is obtained from *seqs* and the reference base call and major variant of the SNP position from *snp_data*. Flanking sequence requirements (size of flanking sequence, requirement of flanking sequence from one or both sides of the SNP) are dependent on application and the algorithm is optimized for the design of Illumina Inc.'s Infinium SNP array which requires 60 bp on either side of the SNP. Left and right flanking sequences are extracted separately; if the SNP is less than the proposed flanking sequence length from either end of the sequence, only the sequence between the end of the reference and the SNP is extracted. The final flanking sequence format, as required by Illumina, combines the left and right flanking sequences by placing a string (left square bracket followed by the reference base, a slash, the SNP base, and a right square bracket) representing the reference and SNP bases between them (Figure 4.4(b)). If both the left and right flanking sequences were less than 60 bp or the reference base was an International Union of Pure and Applied Chemistry (IUPAC) ambiguity code (an IUPAC code indicating more than one nucleotide, Appendix A) the flanking sequence for this position is stored as N/A.

For many applications it is important to know if another SNP falls within the flanking sequence of the current SNP. For this reason, SNPs in the flanking sequence are converted to their IUPAC ambiguity codes using the SNP and reference alleles. The algorithm iterates over each position in the flanking sequence and converts the flanking sequence position to a position in the reference sequence by adding the position in the

a) 5′ -- ACTGT...CATGCTTAGCT...CTAGC -- 3′

b) 5′ -- ACYGT...CATGC[ref/var]TAGCT...CTAGC -- 3′

Figure 4.4: An example of flanking sequences to the left (5′) and right (3′) of the SNP (highlighted red) (a) before and after formatting (b). Formatting replaces SNPs in the flanking sequence with IUPAC ambiguity codes as well as the nucleotide at the SNP position with both the reference and variant nucleotides, separated by a slash, and surrounded by square brackets. In this example, a T/C SNP occurs at position 3 and is replaced by the appropriate ambiguity code Y (highlighted green).

flanking sequence to the start position of the flanking sequence in the reference. If a SNP exists in *snp_data* at the reference position the IUPAC code, based on the reference base and SNP, is determined and replaces the base in the flanking sequence. Once all the SNPs in the flanking sequence have been converted the formatted string can be stored (Figure 4.4(b)) in the *snp_data* structure.

4.5.5 Outputting SNP Results

The algorithm's final step is to output the collected SNP and alignment data as described in Algorithm 4.5.6. It begins by outputting a header line with the following format: SNP id, reference id, SNP position, flanking sequence, reference base, and then a three part header for each individual (base call, frequency, and total depth). For each reference sequence the algorithm iterates over the SNP positions and generates a unique id for each SNP that consists of the reference id and the SNP position. The elements of the output are collected in two arrays (*parts*, *snp_parts*) to be printed once all the data is processed in the correct order. The first output array (*parts*) stores the following elements: SNP id, reference id, reference position, flanking sequence, and reference base (all pulled from the *snp_data* data structure). Then for each individual (using the order defined by the *output_order* array) the algorithm determines if a SNP call exists. If a SNP was called the following elements are stored in the second output array (*snp_parts*): the SNP call, SNP frequency, and total depth. Otherwise the algorithm determines if there is read coverage for the position or if the position is null. If there is read coverage, all of the base calls are collected for the reference position, along with base call frequencies, and the total depth and placed into *snp_parts*. If no reads were observed covering the reference position, a null is indicated by storing in *snp_parts* three X's; one for the call, one for the frequency, and one for the read depth of the individual. Once data has been collected from all of the individuals, the output can be printed to the $Output\_file$ in a tab delimited format. Figure 4.2 shows an example of the SNP output format including the header line and output of a SNP at position 100 in chromosome 1. As a whole, this pipeline accomplishes step (b) of Figure 4.1 as it has determined appropriate call data (reference, SNP, or null) for each individual, for each reference position in which a SNP has been called in at least one individual.

Algorithm 4.5.1: GEN_INFINIUM_ARRAY_DESIGN($SNP\_dir, Ref\_fasta, SAM\_dir, Out\_file$)

comment: *SNP_dir* - a directory containing all SNP reports (one per individual)

comment: *Ref_fasta* - a FASTA file containing one record for each reference sequence

comment: *SAM_dir* - a directory containing all SAM alignment files (one per individual)

comment: *Out_file* - the name of the file to output results to

main

$SNP\_reports \leftarrow$ list of SNP reports from *SNP_dir*

$(snp\_data, output\_order) \leftarrow$ PROCESS_SNP_REPORTS($SNP\_reports$) (i)

$SAM\_files \leftarrow$ list of SAM files from *SAM_dir*

for each $sam\_file \in SAM\_files$ (ii)

do
- **comment:** call *process_sam_file* on each file, one after the other
- PROCESS_SAM_FILE(sam_file, snp_data)

$seqs \leftarrow$ PARSE_REF_SEQS(Ref_fasta) (iii)

ADD_MAJOR_VARIANT($snp\_data, output\_order$) (iv)

GENERATE_FLANKING_SEQUENCE($seqs, snp\_data$) (v)

OUTPUT_TABLE($snp\_data, output\_order, Out\_file$) (vi)

Algorithm 4.5.2: PROCESS_SNP_REPORTS($SNP\_reports$)

```
variables (data, output_order)
comment: data is a hash
comment: output_order is an array
for each filename ∈ SNP_reports
do { open filename for reading
     ind_name ← individual name from filename
     store ind_name in output_order
     while there are lines in file
     do { variables
          ref_id,
          pos,
          variant_type,
          ref_base,
          variant,
          variant_frequency,
          variant_depth,
          total_depth ← elements of line split on tab
          comment: total_depth is depth of variant + depth of reference
          skip if variant_type != SNP
          if there is more than one variant
          then { variant ← string of variants separated by /
                 variant_frequency ← string of variant frequencies separated by /
          store variant in data at ref_id→pos→ind_name→ variant
          store variant_frequency in data at ref_id→pos→ind_name→ variant_frequency
          store total_depth in data at ref_id→pos→ind_name→ total_depth
return (data, output_order)
```

Algorithm 4.5.3: PROCESS_SAM_FILE(sam_file, snp_data)

open *sam_file* for reading
ind_name ← individual name from *sam_file*
while there are lines in *sam_file*
do
- skip if line does not contain read data
- **variables** $read\_id, ref\_id, start\_pos, cigar\_string, read\_seq$ ← elements of line split on tab
- **comment:** Skip reads from reference sequences that do not have SNPs in them
- skip if ref_id is not in *snp_data*
- **variables** *seq*, *ops*, *op_cnts*
- **comment:** *seq*, *ops*, & *op_cnts* are arrays
- **comment:** Split *read_seq* into an array of individual bases
- *seq* ← *read_seq*
- **comment:** Split *cigar_string* into arrays containing individual operations (*ops*)
- **comment:** and the number of times to apply that operation (*op_cnts*)
- (*ops*, *op_cnts*) ← *cigar_string*
- **comment:** Initialize variables to track the position in both the read and the reference
- $seq\_pos \leftarrow 0$
- $ref\_pos \leftarrow start\_pos$
- **comment:** Process the alignment of the read to the reference
- PROCESS_ALIGNMENT($snp\_data, seq, seq\_pos, ref\_pos, ops, op\_cnts, ind\_name$)

Algorithm 4.5.4: PROCESS_ALIGNMENT($snp\_data, seq, seq\_pos, ref\_pos, ops, op\_cnts, ind\_name$)

```
comment: Loop over ops
for i ← 0 to Number of ops
do { operation ← ops[i]
     if operation == [MX =]
     then { for j ← 1 to op_cnts[i]
            do { base ← seq[seq_pos]
                 skip if base != [ATCG]
                 comment: if there is no SNP for this individual then
                 comment: increment the count for the total read depth and the base
                 if { exists snp_data→ref_id→ref_pos &&
                    { !exists snp_data→ref_id→ref_pos→ind_name→snp
                   then { snp_data→ref_id→ref_pos→ind_name→depth++        (i)
                        { snp_data→ref_id→ref_pos→ind_name→base++         (ii)
                 comment: Increment the seq_pos and ref_pos
                 seq_pos ← seq_pos + 1
                 ref_pos ← ref_pos + 1
     else if operation == D
     then { for j ← 1 to op_cnts[i]
            do { base ← *
                 comment: If there is no SNP for this individual then
                 comment: increment the count for the total read depth and the base
                 if { exists snp_data→ref_id→ref_pos &&
                    { !exists snp_data→ref_id→ref_pos→ind_name→snp
                   then { snp_data→ref_id→ref_pos→ind_name→depth++        (iii)
                        { snp_data→ref_id→ref_pos→ind_name→base++         (iv)
                 comment: Increment only the ref_pos when there is a deletion in the read
                 ref_pos ← ref_pos + 1
     else if operation == I || operation == S
     then { comment: Operation I indicates an insertion to the reference
          { comment: Operation S indicates soft clipping of the read
          { comment: These cases require only the position in the read to change
          { seq_pos ← seq_pos + op_counts[i]
     else if operation == N
     then { comment: Operation N indicates a skipped region from the reference
          { comment: This cases require only the position in the reference to change
          { ref_pos ← ref_pos + op_counts[i]
```

Algorithm 4.5.5: GENERATE_FLANKING_SEQUENCES($seqs$, snp_data)

```
for each ref_id ∈ keys snp_data
do { for each ref_pos ∈ keys snp_data→ref_id
     do { ref_seq ← seqs→ref_id
          ref ← snp_data→ref_id→ref_pos→ref
          major_variant ← snp_data→ref_id→ref_pos→major_variant
          left_coord ← ref_pos - 61
          left_size ← 60
          comment: These are based on Illumina Infinium flanking requirements
          if left_coord < 0
            then { left_coord ← 0
                   left_size ← ref_pos - 1
          lf_seq ← substring of ref_seq starting at left_coord extracting left_size bases
          rf_seq ← substring of ref_seq starting at ref_pos extracting 60 bases
          if ref != [ATCG] || ( length(lf_seq) < 60 && length(rf_seq) < 60)
            then { snp_data→ref_id→ref_pos→flanking ← "N/A"
                   skip to next ref_pos
          comment: SNPs that occur in the flanking sequence of another SNP
          comment: must be converted to IUPAC ambiguity code
          len ← the greater of length(lf_seq) || length(rf_seq)
          for pos ← 1 to len
          do { lf_pos ← ref_pos - pos
               rf_pos ← ref_pos + pos
               comment: Operate on the left flanking sequence
               if exists snp_data→ref_id→lf_pos && pos <= length(lf_seq)
                 then { flanking_ref ← snp_data→ref_id→lf_pos→ref
                        flanking_var ← snp_data→ref_id→lf_pos→major_variant
                        flanking_snp ← IUPAC ambiguity code for flanking_ref/flanking_var
                        replace base call at SNP with IUPAC code
               comment: Operate on the right flanking sequence
               if exists snp_data→ref_id→rf_pos && pos <= length(rf_seq)
                 then { flanking_ref ← snp_data→ref_id→rf_pos→ref
                        flanking_var ← snp_data→ref_id→rf_pos→major_variant
                        flanking_snp ← IUPAC ambiguity code for flanking_ref/flanking_var
                        replace base call at SNP with IUPAC code
          snp_data→ref_id→rf_pos→flanking ← lf_seq . "[ref/major_variant]" . rf_seq
```

Algorithm 4.5.6: OUTPUT_TABLE(*snp_data*, *output_order*, *Out_file*)

```
open Out_file for writing
comment: Print a header line to the output file (tab delimited)
print to Out_file SNP id, Reference id, SNP position, Flanking sequence, Reference Base
for each ind_name ∈ output_order
  do {print ind_name Base call, ind_name Frequency, ind_name Total depth
for each ref_id ∈ keys snp_data
do { for each ref_pos ∈ keys snp_data→ref_id
     do { variables parts, snp_parts
          comment: parts and snp_parts are arrays
          snp_id ← ref_id-ref_pos
          ref_base ← snp_data→ref_id→ref_pos→ref
          skip if ref_base != [ATCG]
          store ref_base in snp_parts
          store snp_id, ref_id, ref_pos, snp_data→ref_id→ref_pos→flanking in parts
          for each ind_name ∈ output_order
          do { if exists snp_data→ref_id→ref_pos→ind_name→snp
               then { comment: If there was a SNP called in the line
                      store snp_data→ref_id→ref_pos→ind_name→snp in snp_parts
                      store snp_data→ref_id→ref_pos→ind_name→frequency in snp_parts
                      store snp_data→ref_id→ref_pos→ind_name→total_depth in snp_parts
               else { comment: Check to see if there were reads covering ref_pos
                      total_depth ← snp_data→ref_id→ref_pos→ind_name→depth
                      if total_depth exisits && is > 0
                      then { variables tmp_bases, tmp_counts
                             comment: tmp_bases and tmp_counts are arrays
                             for each base ∈ snp_data→ref_id→ref_pos→ind_name
                             do { comment: collect all base calls
                                  skip if base == 'depth'
                                  if base == ref_base
                                    then base ← 0
                                  count ← snp_data→ref_id→ref_pos→ind_name→base
                                  store base in tmp_bases, count in tmp_counts
                             if size of tmp_bases > 1
                               then { store string of bases separated by / in snp_parts
                                      store string of counts separated by / in snp_parts
                               else store tmp_bases[0], tmp_counts[0] in snp_parts
                             store total_depth in snp_parts
                      else { comment: Put X's into snp_parts to indicate no coverage
                             store X, X, X in snp_parts
          comment: Print a new line of output to Out_file (tab delimited)
          print to Out_file parts, snp_parts
```

4.6 Parallel SAM Processing Algorithms

As next generation sequencing became less expensive, it became more affordable for researchers to generate sequence data at greater depth, from larger numbers of individuals, and in species with larger genomes. These factors combine to dramatically increase the size of the alignment input that is typically used. Figure 4.5 illustrates the increase in alignment input size is linear in the reference size multiplied by the sequence coverage multiplied by the number of individuals, showing the potential for large input sizes with next generation sequencing techniques. For the algorithm described in Section 4.5, the majority of the time taken is for processing of the alignment data (this is Algorithm 4.5.4). Indeed, this step takes an average of 92% of the serial algorithm's processing time for the eight test inputs in Table 4.2. Thus, it became evident that increases in the input size would result in significantly longer running times for our serial algorithm (discussed further in Section 4.9.1). For this reason we began development of a parallelized version of the serial algorithm described in Section 4.5. As the running time of the algorithm is dominated by the time required to process the read alignments (SAM file) for each individual (Table 4.2), the focus is on parallelizing this particular algorithm phase. As the parallel algorithms were designed to calculate the identical result as the serial algorithm, no comparison of the accuracy between the serial and parallel algorithms is required.

Results presented in Table 4.2 were collected on an early 2011 Apple MacBook Pro (Mac OS X version 10.9.1) with a 2.3 GHz Intel Core i7 quad core processor with 16GB of RAM. The Intel processor supports the Intel Hyper-Threading protocol, which allows two CPU threads to run simultaneously per CPU core, resulting in a maximum of 8 CPU threads to execute concurrently.

4.6.1 Parallelization Using Threads

Two algorithms for parallelization of the SAM processing phase are developed which use multiple threads to simultaneously process multiple SAM files from multiple individuals. The parallel 1 algorithm stores only the alignment positions for which SNPs have been identified while the parallel 2 algorithm stores alignment data for every base pair of the reference sequence(s). These algorithms use a semaphore variable (*thread_count*) which is shared across threads to limit the number of active processors to a user defined number (default of 8). A default value of eight is used as many modern computers come equipped with quad-core processors, each of which can handle two parallel threads, resulting in eight total CPU threads. However, this value can be set to just the number of individuals (if the number of CPU threads available is greater than the number of individuals) or perhaps more commonly, to the maximum number of CPU threads the user has available.

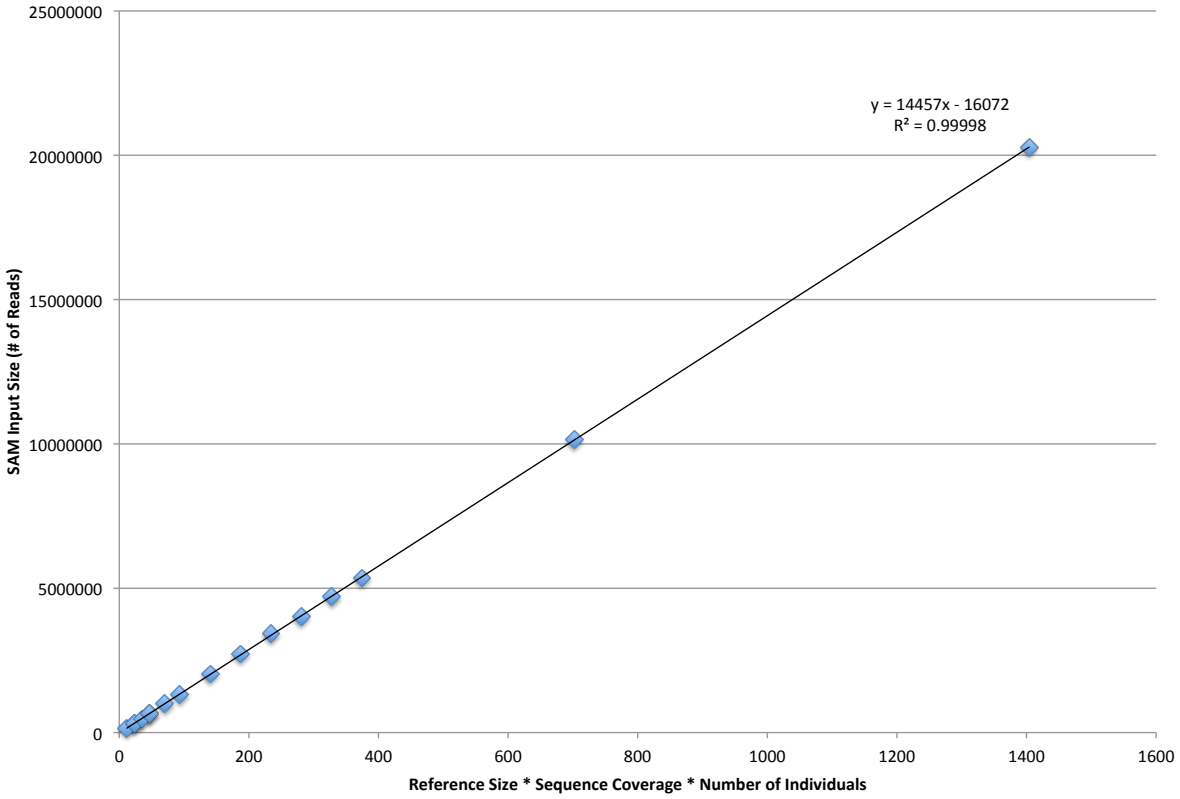

Figure 4.5: Plotting SAM input size (total number of aligned reads across all individuals) for variations in genome size, sequence coverage and number of individuals (Table 4.1) shows the potential for large input data sets with next generation sequencing methods.

During the generation of threads (Algorithm 4.6.1 (iii) & Algorithm 4.6.4 (iii)) *thread_count* is decreased. When enough threads have been created to decrease *thread_count* to zero, new thread generation will be blocked until a running thread increases the value of *thread_count* just before terminating (Algorithm 4.6.2 (iii) & Algorithm 4.6.5 (iii)). Pseudocode has been given where the algorithms differ from that described in Algorithms 4.5.1 - 4.5.6, with numbered statements indicating changes in logic.

Number of Individuals	Reference Size (Mb)	Sequence Coverage (X)	Reference Size * Sequence Coverage * Number of Individuals	SAM size (# of reads)
1	1.17	10	11.7	153140
2	1.17	10	23.4	306995
1	2.34	10	23.4	326807
3	1.17	10	35.1	469354
2	2.34	10	46.8	658311
1	4.68	10	46.8	660054
3	2.34	10	70.2	1005260
2	4.68	10	93.6	1329276
3	4.68	10	140.4	2028043
4	4.68	10	187.2	2724822
5	4.68	10	234	3422310
6	4.68	10	280.8	4009461
7	4.68	10	327.6	4722103
8	4.68	10	374.4	5352694
3	4.68	50	702	10141668
3	4.68	100	1404	20280932

Table 4.1: The size of the SAM input is a function of the number of individuals, the size of the reference sequence set (Mb), and the sequence coverage (fold). The sum of the aligned reads across all SAM files is given as a representation of the total SAM input size for variations in each of the three variables.

Number of Individuals	Reference Size (Mb)	Sequence Coverage (X)	SAM Processing Time (sec)	Total Time (sec)	% of Running Time for SAM Processing
1	4.68	10	34.67	38.67	89.66
2	4.68	10	70.67	76.33	92.58
3	4.68	10	105.67	114	92.69
4	4.68	10	142.33	153	93.03
5	4.68	10	181.33	194	93.47
6	4.68	10	210.67	229.67	91.73
7	4.68	10	253.33	274.33	92.35
8	4.68	10	297.33	321.67	92.44

Table 4.2: Percentage of running time required for SAM processing, for different numbers of individuals (SAM files), of the Serial SAM processing algorithm (Section 4.5). SAM processing and total running time results are in seconds and are averaged over 3 replicates per number of individuals.

4.6.2 Parallel 1 Algorithm

In the first parallel algorithm, processing of SAM files (Algorithms 4.6.2 & 4.6.3) is essentially the same as in Section 4.5, where alignment data is stored only for positions with SNPs identified at that position in one or more individuals. However, alignment results are stored in a local variable in each parallel thread (Algorithm 4.6.2 (i) & Algorithm 4.6.3 (i - iv)) and then returned (Algorithm 4.6.2 (iii)). In order to store only alignment positions with SNP data, the SNP reports are processed first and the resulting data structure passed to each thread. This algorithm merges thread results (alignment data) with the existing *snp_data* data structure (Algorithm 4.6.1 (iv)). Since the merged *snp_data* data structure in this algorithm is the same as the *snp_data* data structure in Section 4.5, there were no changes made to the algorithm for outputting the results (Algorithm 4.5.6).

Section 4.7 will evaluate this algorithm's computational complexity with respect to time and space, while Section 4.9 will evaluate real-world time and memory requirements based on a Perl implementation, and Section 4.11 will discuss the advantages/disadvantages of the implementation of this algorithm compared to the implementations of other algorithms in this work.

Algorithm 4.6.1: GEN_INFINIUM_ARRAY_DESIGN($SNP\_dir, Ref\_fasta, SAM\_dir, Out\_file, max\_threads$)

comment: *SNP_dir* - a directory containing all SNP reports (one per individual)

comment: *Ref_fasta* - a FASTA file containing one record for each reference sequence

comment: *SAM_dir* - a directory containing all SAM alignment files (one per individual)

comment: *Out_file* - the name of the file to output results to

main

$SNP\_reports \leftarrow$ list of SNP reports from *SNP_dir*
$(snp\_data, output\_order) \leftarrow$ PROCESS_SNP_REPORTS($SNP\_reports$)
$SAM\_files \leftarrow$ list of SAM files from *SAM_dir*
variables *threads*, *sam_data* (i)
comment: *threads* is an array, *sam_data* is a hash
$thread\_count \leftarrow 8$ (ii)
comment: *thread_count* is a semaphore (variable shared by all threads)
comment: the value *thread_count* gets is the total number of threads to start at once
for each $sam\_file \in SAM\_files$ (iii)
do
- **comment:** decrease *thread_count*
- **comment:** when *thread_count* reaches 0 it blocks creation of new threads
- *thread_count*–
- **comment:** Generate a new thread for parsing each SAM file
- $thread \leftarrow$ PROCESS_SAM_FILE($sam\_file, snp\_data, thread\_count$)
- store *thread* in *threads*

for each $thread \in threads$ (iv)
do
- **comment:** Collect the values returned by the parallel thread
- $(ind\_name, data) \leftarrow thread$
- **comment:** Merge the results of the thread with *snp_data*
- **for each** $ref \in keysdata$
  - **do**
    - **for each** *pos* ∈ keys *data*→*ref*
      - **do** {*snp_data*→*ref*→*pos*→*ind_name* ← *data*→*ref*→*pos*

$seqs \leftarrow$ PARSE_REF_SEQS(Ref_fasta)
ADD_MAJOR_VARIANT($snp\_data, output\_order$)
GENERATE_FLANKING_SEQUENCE($seqs, snp\_data$)
OUTPUT_TABLE($snp\_data, sam\_data, output\_order, output\_file$)

Algorithm 4.6.2: PROCESS_SAM_FILE($sam\_file, snp\_data, thread\_count$)

```
open sam_file
ind_name ← individual name from sam_file
variables sam_data                                                    (i)
comment: sam_data is a hash

while there are lines in sam_file
do { skip if line does not contain read data
     variables read_id, ref_id, start_pos, cigar_string, read_seq ← elements of line split on tab
     variables seq, ops, op_cnts
     comment: seq, ops, & op_cnts are arrays
     comment: Split read_seq into an array of individual bases
     seq ← read_seq
     comment: Split cigar_string into arrays containing individual operations (ops)
     comment: and the number of times to apply that operation (op_cnts)
     (ops, op_cnts) ← cigar_string
     seq_pos ← 0
     ref_pos ← start_pos
     PROCESS_ALIGNMENT(snp_data, seq, seq_pos, ref_pos, ops, op_cnts, sam_data, ind_name)
comment: Increase thread_count
comment: when the thread finishes to allow for new threads to be started
thread_count++                                                        (ii)
return (ind_name, sam_data)                                           (iii)
```

Algorithm 4.6.3: PROCESS_ALIGNMENT($snp\_data, seq, seq\_pos, ref\_pos, ops, op\_cnts, sam\_data, ind\_name$)

comment: Loop over *ops*

for $i \leftarrow 0$ **to** Number of ops

do
- $operation \leftarrow ops[i]$
- **if** $operation == [MX =]$
  - **then**
    - **for** $j \leftarrow 1$ **to** $op\_cnts[i]$
      - **do**
        - $base \leftarrow seq[seq\_pos]$
        - skip if *base* != [ATCG]
        - **comment:** if there is no SNP for this individual then
        - **comment:** increment the count for the total read depth and the base
        - **if** exists *snp_data*→*ref_id*→*ref_pos* && !exists *snp_data*→*ref_id*→*ref_pos*→*ind_name*→snp
          - **then**
            - **comment:** Increment the value for total depth and *base*
            - *sam_data*→*ref_id*→*ref_pos*→total++ (i)
            - *sam_data*→*ref_id*→*ref_pos*→*base*++ (ii)
        - **comment:** Increment the *seq_pos* and *ref_pos*
        - $seq\_pos \leftarrow seq\_pos++$
        - $ref\_pos \leftarrow ref\_pos++$
- **else if** $operation == D$
  - **then**
    - **for** $j \leftarrow 1$ **to** $op\_cnts[i]$
      - **do**
        - $base \leftarrow *$
        - **comment:** If there is no SNP for this individual then
        - **comment:** increment the count for the total read depth and the base
        - **if** exists *snp_data*→*ref_id*→*ref_pos* && !exists *snp_data*→*ref_id*→*ref_pos*→*ind_name*→snp
          - **then**
            - **comment:** Increment the value for total depth and *base*
            - *sam_data*→*ref_id*→*ref_pos*→total++ (iii)
            - *sam_data*→*ref_id*→*ref_pos*→*base*++ (iv)
        - **comment:** Increment only the *ref_pos* when there is a deletion in the read
        - $ref\_pos \leftarrow ref\_pos++$
- **else if** $operation == I \parallel operation == S$
  - **then**
    - **comment:** Increase the *seq_pos* by $op\_cnts[i]$ when there is an insertion in the read
    - $seq\_pos \leftarrow seq\_pos + op\_cnts[i]$
- **else if** $operation == N$
  - **then**
    - **comment:** Operation N indicates a skipped region from the reference
    - **comment:** so we increase *ref_pos* by $op\_cnts[i]$ and do not change *seq_pos*
    - $ref\_pos \leftarrow ref\_pos + op\_cnts[i]$

4.6.3 Parallel 2 Algorithm

This algorithm stores alignment data for all possible reference positions and instead of processing the SNP report data first, as in Algorithms 4.5.1 and 4.6.1, the SAM data (*sam_data*) is parsed first (Algorithm 4.6.4 (i)). This algorithm stores in a second data structure, for every alignment position, a count of each alignment element (bases A, T, C, and G plus the gap character (*) and a total depth). Adding a second structure to the algorithm required changes to how reference and null positions are determined during the output of the results (Algorithm 4.6.7). The initial process is the same and iteration occurs over the positions within each reference sequence that have a SNP called in them. The algorithm then iterates over each individual and if a SNP was called then the data contained in *snp_data* is output. If no SNP was called in the individual the algorithm queries the *sam_data* structure containing the bit vectors for the depth of reads available at that reference position (Algorithm 4.6.7 (i)). If there were reads aligned to that reference position in the individual then the count of each reported base at that position (Algorithm 4.6.7 (ii & iii)) and the base frequencies can be determined. These values along with the total depth are then reported for the individual.

Algorithm 4.6.4: GEN_INFINIUM_ARRAY_DESIGN($SNP\_dir, Ref\_fasta, SAM\_dir, Out\_file, max\_threads$)

comment: *SNP_dir* - a directory containing all SNP reports (one per individual)

comment: *Ref_fasta* - a FASTA file containing one record for each reference sequence

comment: *SAM_dir* - a directory containing all SAM alignment files (one per individual)

comment: *Out_file* - the name of the file to output results to

main

$SAM\_files \leftarrow$ list of SAM files from *SAM_dir*

variables *threads*, *sam_data* (i)

comment: *threads* is an array, *sam_data* is a hash

$thread\_count \leftarrow 8$ (ii)

comment: *thread_count* is a semaphore (variable shared by all threads)

comment: the value *thread_count* gets is the total number of threads to start at once

for each $sam\_file \in SAM\_files$ (iii)

do
- **comment:** decrease *thread_count*
- **comment:** when *thread_count* reaches 0 it blocks creation of new threads
- *thread_count*–
- **comment:** Generate a new thread for parsing each SAM file
- $thread \leftarrow$ PROCESS_SAM_FILE($sam\_file, thread\_count$)
- store *thread* in *threads*

for each $thread \in threads$ (iv)

do
- **comment:** Collect the values returned by the parallel thread
- $(ind\_name, data) \leftarrow thread$
- **comment:** *data* is a hash with format *ref_pos*→*base* where base is A, T, C, G, * or total
- store *data* in *sam_data* at *ind_name* (v)
- **comment:** structure therefore becomes *ind_name*→*ref_pos*→*base*

$SNP\_reports \leftarrow$ list of SNP reports from *SNP_dir* (vi)

$(snp\_data, output\_order) \leftarrow$ PROCESS_SNP_REPORTS($SNP\_reports$)

$seqs \leftarrow$ PARSE_REF_SEQS(Ref_fasta)

ADD_MAJOR_VARIANT($snp\_data, output\_order$)

GENERATE_FLANKING_SEQUENCE($seqs, snp\_data$)

OUTPUT_TABLE($snp\_data, sam\_data, output\_order, output\_file$)

Algorithm 4.6.5: PROCESS_SAM_FILE($sam\_file, thread\_count$)

open *sam_file*
ind_name ← individual name from *sam_file*
variables *sam_data* (i)
comment: *sam_data* is a hash

while there are lines in *sam_file*
do
- skip if line does not contain read data
- **variables** $read\_id, ref\_id, start\_pos, cigar\_string, read\_seq$ ← elements of line split on tab
- **variables** *seq*, *ops*, *op_cnts*
- **comment:** *seq*, *ops*, & *op_cnts* are arrays
- **comment:** Split *read_seq* into an array of individual bases
- *seq* ← *read_seq*
- **comment:** Split *cigar_string* into arrays containing individual operations (*ops*)
- **comment:** and the number of times to apply that operation (*op_cnts*)
- (*ops*, *op_cnts*) ← *cigar_string*
- *seq_pos* ← 0
- ref_pos ← *start_pos*
- **comment:** Initialize an empty data structure for the total depth if it does not exist
- **if** !exists *sam_data*→*ref_id*→total
  then *sam_data*→*ref_id*→total = ' ' (ii)
- PROCESS_ALIGNMENT(*seq*, *seq_pos*, ref_pos, *ops*, *op_cnts*, *sam_data*, *ind_name*)

comment: Increase *thread_count*

comment: when the thread finishes to allow for new threads to be started

thread_count++ (iii)
return (*ind_name*, *sam_data*) (iv)

Algorithm 4.6.6: PROCESS_ALIGNMENT($seq, seq\_pos, ref\_pos, ops, op\_cnts, sam\_data, ind\_name$)

```
comment: Loop over ops
for i ← 0 to Number of ops
do { operation ← ops[i]
     if operation == [MX =]
       then { for j ← 1 to op_cnts[i]
              do { base ← seq[seq_pos]
                   skip if base != [ATCG]
                   comment: Initialize an empty data structure for base
                   if !exists sam_data→ref_id→base
                     then sam_data→ref_id→base = ' '                          (i)
                   comment: Increment the value for total depth and base
                   (sam_data→ref_id→total, ref_pos, 16)++                     (ii)
                   (sam_data→ref_id→base, ref_pos, 16)++                      (iii)
                   comment: Increment the seq_pos and ref_pos
                   seq_pos ← seq_pos + 1
                   ref_pos ← ref_pos + 1 }
     else if operation == D
       then { for j ← 1 to op_cnts[i]
              do { base ← *
                   comment: Initialize an empty data string for deletions
                   if !exists sam_data→ref_id→base
                     then sam_data→ref_id→base = ' '                          (iv)
                   comment: Increment the value for total depth and base
                   (sam_data→ref_id→total, ref_pos, 16)++                     (v)
                   (sam_data→ref_id→base, ref_pos, 16)++                      (vi)
                   comment: Increment only the ref_pos when there is a deletion in the read
                   ref_pos ← ref_pos + 1 }
     else if operation == I || operation == S
       then { comment: Increase the seq_pos by op_cnts[i] when there is an insertion in the read
              seq_pos ← seq_pos + op_cnts[i] }
     else if operation == N
       then { comment: Operation N indicates a skipped region from the reference
              comment: so we increase ref_pos by op_cnts[i] and do not change seq_pos
              ref_pos ← ref_pos + op_cnts[i] }
```

Algorithm 4.6.7: OUTPUT_TABLE($snp\_data, sam\_data, output\_order, Out\_file$)

```
open Out_file for writing
print to Out_file SNP id, Reference id, SNP position, Flanking sequence, Reference Base
for each ind_name ∈ output_order
  do {print ind_name Base call, ind_name Frequency, ind_name Total depth
for each ref_id ∈ keys snp_data
do { for each ref_pos ∈ keys snp_data→ref_id
     do { variables parts, snp_parts
          snp_id ← ref_id-ref_pos
          ref_base ← snp_data→ref_id→ref_pos→ref
          skip if ref_base != [ATCG]
          store ref_base in snp_parts
          store snp_id, ref_id, ref_pos, snp_data→ref_id→ref_pos→flanking in parts
          for each ind_name ∈ output_order
          do { if exists snp_data→ref_id→ref_pos→ind_name→snp
               then { store snp_data→ref_id→ref_pos→ind_name→snp in snp_parts
                      store snp_data→ref_id→ref_pos→ind_name→frequency in snp_parts
                      store snp_data→ref_id→ref_pos→ind_name→total_depth in snp_parts
               else { comment: Check to see if there were reads covering ref_pos
                      total_depth ← (sam_data→ind_name→ref_id→depth, ref_pos, 16)   (i)
                      if total_depth exisits && is > 0
                      then { variables tmp_bases, tmp_counts
                             for each base ∈ sam_data→ind_name→ref_id
                             do { skip if base == 'depth'
                                  if base == ref_base
                                    then base ← 0
                                  vec_string ← sam_data→ind_name→ref_id→base   (ii)
                                  count ← (vec_string, ref_pos, 16)            (iii)
                                  store base in tmp_bases, count in tmp_counts
                             if size of tmp_bases > 1
                               then { store string of bases separated by / in snp_parts
                                      store string of counts separated by / in snp_parts
                               else store tmp_bases[0], tmp_counts[0] in snp_parts
                             store total_depth in snp_parts
                      else {store X, X, X in snp_parts
          print to Out_file parts, snp_parts
```

4.7 Evaluation Of Computational Complexity

The generation of the table of SNP data across all of the sequenced individuals (as described in Section 4.2) is the most computationally intensive step in the process of generating SNP data with sufficient biological information to proceed with the development of a SNP genotyping array in complex organisms. For this reason we have evaluated, in Θ-notation (Section 2.5.8), the algorithms described in the previous sections for their computational complexity.

4.7.1 Time Complexity

The algorithms we proposed in Sections 4.5, 4.6.2, and 4.6.3 perform the same function. The SNP reports are processed and stored in a structure (*snp_data*), as are the SAM alignment files (stored in *snp_data* in Algorithms 4.5.1 and 4.6.1 and in *sam_data* in Algorithm 4.6.4). We parse the reference sequences, add the major variant and flanking sequence for each SNP and format and output the results from all individuals into a table. As the serial algorithm described in Section 4.5 is fundamentally only different from the algorithms in Sections 4.6.2 and 4.6.3 during the processing of the SAM alignment files, the time complexity in Θ-notation of the common algorithm phases will be described first and time complexity of processing SAM files for the serial algorithm and parallel algorithms will be described separately.

SNP Processing

The algorithms described previously loop over the number of individuals (n) and for each individual, opens the SNP report and processes all the lines of the file (s) (the number of lines for an individual i is denoted by s_i). As such, the processing of the SNP reports (Algorithm 4.5.2) in the algorithms requires time equal to the sum of the number of lines across all SNP reports. This results in a time complexity for SNP processing of $\Theta(R)$, where

$$R = \sum_{i=1}^{n} s_i. \tag{4.1}$$

Reference Parsing

The algorithm for parsing the reference sequences simply passes over each line of the reference sequence file (l) requiring time $\Theta(l)$.

Addition of major variant

The addition of the major variant requires looping over the number of reference sequences with variants in them (r) and for each reference looping over the number of variants (v_i for reference i) found in that sequence, represented by

$$V = \sum_{i=1}^{r} v_i. \tag{4.2}$$

The algorithm then looks at the variant in each of the individuals with a SNP call (at most n), resulting in a time requirement of $\Theta(V \cdot n)$.

Generation Of Flanking Sequences

Generation of flanking sequences (Algorithm 4.5.5) again requires looping over the reference sequences with SNPs (r) and then the variants (v_i) (Equation 4.2). Then for each of the variants, an iteration over the length of the flanking sequence (f) occurs, resulting in a time complexity of $\Theta(V \cdot f)$.

Outputting Results

Finally the algorithms take each reference sequence (r) and for every variant position (v_i) (Equation 4.2), loops over the individuals (n) and collects and prints the data to the output file (Algorithm 4.5.6). This process takes time $\Theta(V \cdot n)$.

Serial SAM Processing

The processing of SAM files (Algorithms 4.5.3 & 4.5.4) loops over the number of individuals (n) and for each individual opens the alignment file. Alignment files contain a number of header lines (h) that do not need to be processed and a number of alignment lines (a). For each alignment line, the length of the alignment, which is made up of a number of operations (o) where each operation has an associated count (c_i), must be traversed. Since the length of the alignment can be determined by

$$L = \sum_{i=1}^{o} c_i, \tag{4.3}$$

the total length of the aligned reads can be represented as $a \cdot L$. The effects of header and alignment lines can then be combined across individuals using

$$S = \sum_{i=1}^{n} (h_i + (a_i \cdot L)), \tag{4.4}$$

the resulting time complexity of the SAM processing phase of the serial algorithm is $\Theta(S)$.

Parallel SAM Processing

In the parallel SAM processing algorithms (Algorithms 4.6.2 & 4.6.3, 4.6.5 & 4.6.6) processing of the reads from each SAM file requires the same time, $\Theta(h + (a \cdot L))$, as in the serial algorithm (Algorithms 4.5.3 & 4.5.4). However, using multi-threaded computing, the processing of each distinct individual in Equation 4.4 is independent from the others. Parallelization requires the distribution of jobs (processing of individuals) to CPU threads. To begin, let n be the number of individuals in the input, t be the number of available threads for processing data, and T_j be the time to process an individual SAM file, where T_j is proportional to $h_i + (a_i \cdot L)$. Then two cases for the time complexity of parallel processing SAM data can be identified as follows:

1. The number of individuals is less than the number of threads available ($n \leq t$).

2. The number of individuals is greater than the number of threads available ($n > t$).

For case 1 the function

$$P = \max_{1 \leq j \leq n} (T_j), \tag{4.5}$$

represents the longest processing time of any SAM file in the input and since each individual receives its own thread for processing this is the maximum time required for SAM processing. Therefore, the time complexity of case 1 is $\Theta(P)$.

In case 2, if the size of the SAM file for each individual is the same, then distribution of jobs has no affect on the time complexity. Therefore the function

$$\lceil n/t \rceil, \tag{4.6}$$

can be substituted for the upper bound of summation in Equation 4.4 resulting in a time complexity for processing SAM data for the parallel algorithms of $\Theta(P')$, where

$$P' = \sum_{i=1}^{\lceil n/t \rceil} (h_i + (a_i \cdot L)). \tag{4.7}$$

When the sizes of the SAM file vary from individual to individual, the parallel algorithms process SAM files based on the input order. Although the optimal partition of jobs to threads is not calculated, as soon as a thread terminates, if there are individuals left to process, a new thread is created. Therefore, a lower bound on the time complexity of the parallel algorithms would be $\Omega(S/t)$. The implementation in Section 4.8 and the performance profiling of that implementation in Section 4.9 will test how close the time is to the optimal speedup.

Overall Time Complexity

By combining the terms from all of the algorithm phases we get an overall complexity for running time of $\Theta(R + l + (V \cdot n) + (V \cdot f) + S)$ for the serial algorithm. To get the complexity of the parallel algorithms we simply divide the term for the serial SAM processing (S) with the number of threads (t) to give $\Omega(R + l + (V \cdot n) + (V \cdot f) + S/t)$. The dominating factor of both the serial and parallel algorithms is that of the SAM processing (S and S/t, respectively). As such, the overall time complexity of the algorithms can be represented by $\mathcal{O}(S)$.

Table 4.3 summarizes the time complexity, of each algorithm phase for both the serial and parallel algorithms. Although some terms of the complexity functions are multiplicative, the terms are all less than the size of the input. Therefore, the parallel 1 and parallel 2 algorithms are expected to increase linearly with respect to their inputs.

4.7.2 Space Complexity

Based on the implementations of the serial and parallel 1 algorithms there are two main requirements for space, a data structure which contains information for each individual at each SNP position found in the reference sequences and a data structure which contains the reference sequence data itself. The SNP data structure contains the union of all SNP positions across all reference sequences (known as the unique SNP positions) (Equation 4.2, V) multiplied by the number of individuals (n). The space required for the reference sequence data structure is based on the total number of base pairs in the reference (b). Therefore the overall

Algorithm Phase	Serial Algorithm	Parallel Algorithm(s)
SNP report processing	$\Theta(R)$, $R = \sum_{i=1}^{n} s_i$	-
SAM file processing	$\Theta(S)$, $S = \sum_{i=1}^{n} (h_i + (a_i \cdot L))$, $L = \sum_{i=1}^{o} c_i$	$\Omega(S/t)$
Reference sequence parsing	$\Theta(l)$	-
Addition of the major variant	$\Theta(V \cdot n)$, $V = \sum_{i=1}^{r} v_i$	-
Flanking sequence generation	$\Theta(V \cdot f)$	-
Output of results	$\Theta(V \cdot n)$	-
Overall	$\Theta(R + l + (V \cdot n) + (V \cdot f) + S)$	$\Omega(R + l + 2(V \cdot n) + (V \cdot f) + S/t)$

Table 4.3: The time complexity of each algorithm phase of the serial and parallel algorithms described in Section 4.2. A dash in the parallel algorithms column indicates that the complexity is the same as for the serial algorithm. The terms of the complexity functions are as follows: n is the number of individuals, s is the number lines in the SNP report, o is the number of cigar operations, c is the number of times to apply operation, h is the number of header lines and a is the number of alignment lines, t is the maximum number threads, l is the number of lines in the reference sequence FASTA file, r is the number of references with SNPs, v is the number of variant positions in the reference, and f is the length of the flanking sequence.

space complexity of the serial algorithms is $\Theta((V \cdot n) + b)$. The space requirements of each of the three algorithms will be experimentally determined using the implementation in Section 4.8 and the performance profiling of that implementation in Section 4.9.

4.8 Implementation Of Algorithms In Perl

The algorithms serial (Section 4.5), parallel 1 (Section 4.6.2), and parallel 2 (Section 4.6.3) are implemented in the Perl programming language based on the pseudocode provided in each section. Perl was selected as the language of implementation as it provides quick development time and is a common programming language for many bioinformatics tools. Implementations in other languages are beyond the scope of this thesis and will be discussed in Chapter 5. The two parallelized algorithms are implemented using the Perl threads and Threads::Semaphore modules. The threads module allows a new thread to be created to process each SAM file and the Threads::Semaphore module provides a shared variable across all threads that can be used to manage the number of parallel operations permitted at one time.

As the parallel 2 algorithm requires a second data structure to store all of the alignment data, an evaluation of several structures to store the alignment data is performed using the Perl module Devel::Size [111]. Devel::Size has a method called *total_size* which will report, in bytes, the size of a data structure — including both the elements of the structure and the structure's contents. Table 4.4 gives the size (converted to MB) reported by the *total_size* method, for storing depth per alignment element for each alignment position from a single SAM file (containing 698,767 reads) of the different data structures evaluated. The results of evaluating the structures clearly show that the hash of bit vectors (using 16 bits per element of the bit vector) is the most memory efficient, as such the algorithm is implemented using this method to store the alignment data.

Data Structure	Structure Size (MB)
Hash	1,645.8
Array of Hashes	1,520.8
Array of Arrays	992.4
Hash of Arrays	789.0
Hash of Bit Vectors	78.4

Table 4.4: Reported size, using the *total_size* method of the Perl module Devel::Size, of different data structures for storing alignment data. Each structure was populated with the data from the same SAM file, which contained 698,767 aligned reads.

In this algorithm, each individual has a set of bit vectors, one per alignment element, for each reference sequence (Algorithm 4.6.5 (ii) & Algorithm 4.6.6 (i & iv)). These bit vectors can be stored in a three level Perl hash using the individual name as the key for the first level, the reference sequence id as the key for the second level and the alignment element as the key for the third (Algorithm 4.6.4 (v)). Since the elements of a bit vector are used to represent the positions in the reference, a particular bit vector element represents the count of reads that have the alignment element corresponding to the bit vectors key at the elements position in the reference (Algorithm 4.6.7 (i, i & iii). For example, after parsing `r001` from Figure 4.3 (a) the key `Ref` would point to six bit vectors (one each for A, T, C, G, * and total). To determine the number of reads with base A at position 7 the element at position 7 of the `Ref`→A bit vector would be accessed, returning a count of 1. This structure gives direct access to the alignment information stored in the bit vector at the element representing the reference position of the SNP. Direct access is important for fast querying of the data structure, an important performance aspect due to the potential to query multiple individuals across large numbers of potential SNPs.

4.9 Performance Profiling

In order to compare the performance of the serial and parallel algorithms, the observed running time and memory of each algorithm's Perl implementation was profiled based on different input parameters. This provides a practical assessment of each of the algorithms and allows for predictions as to the limitations of the algorithms and for an assessment of necessary hardware to process desired input (Section 4.11). Profiling was performed using publicly available *E.coli* genome data. Test runs of each algorithm were performed using the *E.coli* strain K12:DH10B (Accession #NC_010473) as the reference sequence; simulated reads were generated from the genome sequences of eight *E.coli* strains (Table.4.5) using the open-source software ART (version 1.5.0) [40] at various coverage levels.

E.coli Strain	NCBI Accession #	Individual #
O157	NC_002655.2	1
O55	NC_013941.1	2
SE11	NC_011415.1	3
E24377A	NC_009801.1	4
HS	NC_009800.1	5
REL606	NC_012967.1	6
SMS-3-5	NC_010498.1	7
O127	NC_011601.1	8

Table 4.5: *E.coli* strains for which simulated reads were generated for use in evaluating the performance of our algorithms. The individual number given for each strain is the order in which they were added to the analysis when multiple individuals were processed together.

Three important factors were identified that can affect the performance of the algorithms:

- **Number of individuals to process** - The number of individuals affects the number of SNPs likely to be found (the union of all SNP positions and not the intersection between individuals is evaluated) and the total amount of SNP and alignment data to be processed. Term n of Table 4.3 represents the effect of the number of individuals.

- **Size of reference sequence(s)** - The size of the reference sequence(s) (total number of base pairs) affects the number of possible SNP positions, the size of the SAM alignments (given a fixed sequencing coverage, which we feel more accurately represents sequencing strategies versus a fixed number of reads)

and the number of lines in the reference sequence FASTA file. Therefore, the effect of the size of the reference sequence(s) is represented indirectly in terms s, r, a, and v and directly in term l of Table 4.3.

- **Sequencing coverage** - Coverage represents the average number of reads covering a single nucleotide in the alignment (fold X). For example, coverage of 10X indicates an average of 10 reads aligned to each nucleotide position in the genome. Thus, for the same genome sequence, higher genome coverage indicates more aligned reads and a larger SAM file. Therefore, the effect of genome sequencing coverage is represented by term a of Table 4.3.

These three factors were varied to produce a set of test input cases (Table 4.6) which could be used to evaluate the time and memory usage performance of the serial, parallel 1, and parallel 2 algorithms, described in Sections 4.5, 4.6.2 and 4.6.3 respectively. Each test case was evaluated by independently aligning the simulated reads, at a specific coverage level, from a set of individuals (*E.coli* strains) to the appropriate reference sequence set (K12 reference sequence: whole genome (4.68 megabases (Mb)), half genome (2.34 Mb) and quarter genome (1.17 Mb)) using the CLC Genomics Workbench version 6.5 (CLC Bio Inc, Aarhus, Denmark) *map reads to reference* tool. It is important to note that the estimated coverage levels indicated are based on the coverage of each *E.coli* genome and that the aligned reads often have reduced coverage of the K12 reference sequence set. SNPs were then detected in the resulting alignments using the CLC Genomics Workbench *probabilistic variant detection* tool. For each individual a SAM file was exported for the read alignments and a tab delimited text file exported for the SNP report generated by CLC Genomics Workbench. The resulting SAM files and tab delimited SNP files for each test case, plus the appropriate reference sequence set, were then passed as input parameters to the algorithm being tested. Each test was performed in triplicate for each algorithm and the running time and memory usage recorded. Results from all replicates were then fit to regression models. Profiling was performed using the Apple MacBook Pro discussed in Section 4.6.

4.9.1 Time Profiling

In order to evaluate the performance of each of our algorithms, we first profiled their running time using the previously described (Table 4.6) test cases. Since the running time is dominated by the parsing of alignment data (Table 4.2) and these algorithms differ only in how they process alignment data, we performed a linear regression of time versus the total number of aligned reads for each of the algorithms. The total number of

Number of Individuals	Reference Size (Mb)	Sequence Coverage (X)
1	4.68	10
2	4.68	10
3	4.68	10
4	4.68	10
5	4.68	10
6	4.68	10
7	4.68	10
8	4.68	10
3	2.34	10
3	1.17	10
3	4.68	50
3	4.68	100

Table 4.6: Combination of factors used for each of the test cases used for time and memory usage profiling of algorithms described in Sections 4.5, 4.6.2 & 4.6.3.

aligned reads was used as it allows for the interactions of each of our performance factors to be combined as in

$$\text{total aligned reads} = \sum_{i=1}^{n} (r \cdot s_i), \tag{4.8}$$

where n is the number of individuals, r is the size of the reference in base pairs, and s is the estimated average sequence coverage.

Results of plotting the data from each algorithm and performing the linear regression (Figure 4.6) shows that the parallel 1 algorithm (Section 4.6.2) requires the least running time while, as expected, the serial algorithm (Section 4.5) requires the most. The R^2 value, which is a statistical measure of how closely the data fit to the model [83], of each algorithm is above 0.98 indicating a very good fit of the data points to the regression. This implies that our expectation of linear growth in running time as the number of aligned reads increases is correct.

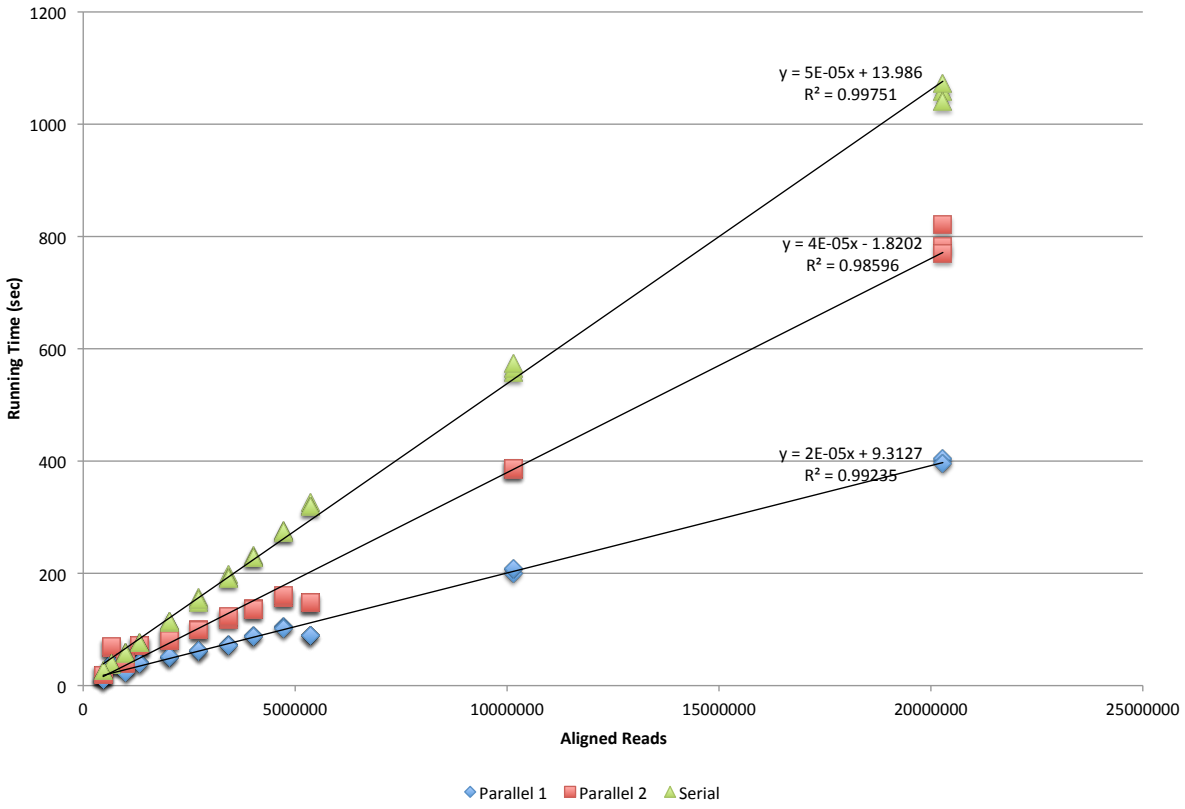

Figure 4.6: Linear regression of time versus aligned reads for the serial, parallel 1, and parallel 2 algorithms, Sections 4.5, 4.6.2 & 4.6.3 respectively. Series data for each of the algorithms can be found in Tables 4.7 (serial), 4.8 (parallel 1), and 4.9 (parallel 2).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Total Aligned Reads	Running Time (sec)
Serial	1	4.68	10X	660054	39
Serial	1	4.68	10X	660054	38
Serial	1	4.68	10X	660054	39
Serial	2	4.68	10X	1329276	76
Serial	2	4.68	10X	1329276	77
Serial	2	4.68	10X	1329276	76
Serial	3	4.68	10X	2028043	113
Serial	3	4.68	10X	2028043	115
Serial	3	4.68	10X	2028043	114
Serial	4	4.68	10X	2724822	148
Serial	4	4.68	10X	2724822	157
Serial	4	4.68	10X	2724822	154
Serial	5	4.68	10X	3422310	198
Serial	5	4.68	10X	3422310	193
Serial	5	4.68	10X	3422310	191
Serial	6	4.68	10X	4009461	232
Serial	6	4.68	10X	4009461	229
Serial	6	4.68	10X	4009461	228
Serial	7	4.68	10X	4722103	276
Serial	7	4.68	10X	4722103	273
Serial	7	4.68	10X	4722103	274
Serial	8	4.68	10X	5352694	326
Serial	8	4.68	10X	5352694	319
Serial	8	4.68	10X	5352694	320
Serial	3	2.34	10X	1005260	59
Serial	3	2.34	10X	1005260	59
Serial	3	2.34	10X	1005260	57
Serial	3	1.17	10X	469354	26
Serial	3	1.17	10X	469354	26
Serial	3	1.17	10X	469354	27
Serial	3	4.68	50X	10141668	557
Serial	3	4.68	50X	10141668	564
Serial	3	4.68	50X	10141668	574
Serial	3	4.68	100X	20280932	1059
Serial	3	4.68	100X	20280932	1073
Serial	3	4.68	100X	20280932	1041

Table 4.7: Raw series data of aligned number of reads and total running time for each of the test cases outlined in Table 4.6 and performed using the serial algorithm described in Section 4.5.

Algorithm	Individuals	Reference Size (Mb)	Coverage	Total Aligned Reads	Running Time (sec)
Parallel 1	1	4.68	10X	660054	39
Parallel 1	1	4.68	10X	660054	39
Parallel 1	1	4.68	10X	660054	38
Parallel 1	2	4.68	10X	1329276	41
Parallel 1	2	4.68	10X	1329276	41
Parallel 1	2	4.68	10X	1329276	40
Parallel 1	3	4.68	10X	2028043	50
Parallel 1	3	4.68	10X	2028043	49
Parallel 1	3	4.68	10X	2028043	52
Parallel 1	4	4.68	10X	2724822	60
Parallel 1	4	4.68	10X	2724822	62
Parallel 1	4	4.68	10X	2724822	62
Parallel 1	5	4.68	10X	3422310	74
Parallel 1	5	4.68	10X	3422310	72
Parallel 1	5	4.68	10X	3422310	72
Parallel 1	6	4.68	10X	4009461	87
Parallel 1	6	4.68	10X	4009461	89
Parallel 1	6	4.68	10X	4009461	88
Parallel 1	7	4.68	10X	4722103	106
Parallel 1	7	4.68	10X	4722103	103
Parallel 1	7	4.68	10X	4722103	101
Parallel 1	8	4.68	10X	5352694	89
Parallel 1	8	4.68	10X	5352694	89
Parallel 1	8	4.68	10X	5352694	89
Parallel 1	3	2.34	10X	1005260	25
Parallel 1	3	2.34	10X	1005260	26
Parallel 1	3	2.34	10X	1005260	24
Parallel 1	3	1.17	10X	469354	11
Parallel 1	3	1.17	10X	469354	11
Parallel 1	3	1.17	10X	469354	11
Parallel 1	3	4.68	50X	10141668	200
Parallel 1	3	4.68	50X	10141668	207
Parallel 1	3	4.68	50X	10141668	208
Parallel 1	3	4.68	100X	20280932	401
Parallel 1	3	4.68	100X	20280932	405
Parallel 1	3	4.68	100X	20280932	396

Table 4.8: Raw series data of aligned number of reads and total running time for each of the test cases outlined in Table 4.6 and performed using the parallel 1 algorithm described in Section 4.6.2.

Algorithm	Individuals	Reference Size (Mb)	Coverage	Total Aligned Reads	Running Time (sec)
Parallel 2	1	4.68	10X	660054	68
Parallel 2	1	4.68	10X	660054	68
Parallel 2	1	4.68	10X	660054	69
Parallel 2	2	4.68	10X	1329276	72
Parallel 2	2	4.68	10X	1329276	71
Parallel 2	2	4.68	10X	1329276	71
Parallel 2	3	4.68	10X	2028043	81
Parallel 2	3	4.68	10X	2028043	83
Parallel 2	3	4.68	10X	2028043	81
Parallel 2	4	4.68	10X	2724822	99
Parallel 2	4	4.68	10X	2724822	100
Parallel 2	4	4.68	10X	2724822	99
Parallel 2	5	4.68	10X	3422310	117
Parallel 2	5	4.68	10X	3422310	121
Parallel 2	5	4.68	10X	3422310	123
Parallel 2	6	4.68	10X	4009461	136
Parallel 2	6	4.68	10X	4009461	137
Parallel 2	6	4.68	10X	4009461	136
Parallel 2	7	4.68	10X	4722103	156
Parallel 2	7	4.68	10X	4722103	156
Parallel 2	7	4.68	10X	4722103	160
Parallel 2	8	4.68	10X	5352694	148
Parallel 2	8	4.68	10X	5352694	148
Parallel 2	8	4.68	10X	5352694	148
Parallel 2	3	2.34	10X	1005260	41
Parallel 2	3	2.34	10X	1005260	41
Parallel 2	3	2.34	10X	1005260	39
Parallel 2	3	1.17	10X	469354	19
Parallel 2	3	1.17	10X	469354	19
Parallel 2	3	1.17	10X	469354	19
Parallel 2	3	4.68	50X	10141668	387
Parallel 2	3	4.68	50X	10141668	384
Parallel 2	3	4.68	50X	10141668	387
Parallel 2	3	4.68	100X	20280932	783
Parallel 2	3	4.68	100X	20280932	769
Parallel 2	3	4.68	100X	20280932	821

Table 4.9: Raw series data of aligned number of reads and total running time for each of the test cases outlined in Table 4.6 and performed using the parallel 2 algorithm described in Section 4.6.3.

The speedup (S), which can be determined using the formula

$$S = \frac{T_{old}}{T_{new}}, \tag{4.9}$$

is a metric for measuring relative performance improvements in computer science [38]. To determine the efficiency of each parallelization method, the speedup of each parallelization algorithm versus the serial algorithm was computed (Table 4.11) and then divided by the theoretical maximum improvement, which is estimated using Amdahl's law (Section 2.5.9). Table 4.10 shows the results of calculating the maximum theoretical speedup using Equation 2.1; values for P are taken from Table 4.2 and the number of processors (N) is the same as the number of individuals. In Table 4.11, the times shown are averages (over three trials) of the SAM alignment processing phase and the efficiency of the parallel algorithms is calculated by dividing the observed speedup by the associated theoretical speedup from Table 4.10 and then multiplying the result by 100. As expected, speedup values increase with the number of CPU threads used, reaching a maximum of 3.38 for the parallel 1 algorithm and 2 for the parallel 2 algorithm at eight CPU threads. However, the efficiency of parallelization decreases as the number of CPU threads increases, resulting in 64.61% efficiency for the parallel 1 algorithm and 38.23% efficiency for the parallel 2 algorithm at eight CPU threads. Maximum parallelization efficiency of both parallel algorithms occurs at two CPU threads, with 108.49% and 57.47% efficiency for the parallel 1 and parallel 2 algorithms respectively. Differences in efficiency between the parallel 1 and parallel 2 algorithms can be attributed to the reduced access speed of bit vectors in Perl compared to the Perl native hash structures. Possible causes for decreased efficiency of the parallel algorithms as the number of CPU threads increases will be discussed in Section 4.10.

N	P	(1 - P)	$S(N)$
2	0.9258	0.0742	1.86
3	0.9269	0.0731	2.62
4	0.9303	0.0697	3.31
5	0.9347	0.0653	3.96
6	0.9173	0.0827	4.24
7	0.9235	0.0765	4.80
8	0.9244	0.0756	5.23

Table 4.10: Theoretical maximum speedup ($S(N)$) as calculated using Equation 2.1 (Amdahl's Law) for 2-8 processors. Values of P are obtained from the percentage of running time required for SAM processing for 2-8 individuals (Table 4.2).

Number of Individuals	Reference Size (Mb)	Coverage	Serial Algorithm Time (S)	Parallel 1 Algorithm Time (P1)	Parallel 2 Algorithm Time (P2)	Parallel 1 Speedup (S/P1)	Parallel 2 Speedup (S/P2)	Parallel 1 Efficiency	Parallel 2 Efficiency
2	4.68	10X	70.67	35	66	2.02	1.07	108.49	57.47
3	4.68	10X	105.67	41.67	73.67	2.54	1.43	97.04	54.64
4	4.68	10X	142.33	49.33	88.67	2.89	1.61	87.36	48.67
5	4.68	10X	181.33	60	108	3.02	1.68	76.18	42.38
6	4.68	10X	210.67	66.33	117	3.18	1.80	74.92	42.41
7	4.68	10X	253.33	79	136	3.21	1.86	66.91	38.77
8	4.68	10X	297.33	88	148.67	3.38	2	64.61	38.23

Table 4.11: Calculation of the speedup and the efficiency (as a percentage of the maximum possibly speedup) of each of the parallel algorithms versus the serial algorithm. Speedup is calculated using Equation 4.9 using average SAM processing time from three trials of each algorithm at each input size. The number of individuals is used as the number of processors in the calculation of the theoretical maximum speedup, since each individual is processed using an independent CPU thread.

4.9.2 Memory Profiling

Since peak memory usage is a limiting factor in the use of the algorithms, the memory usage of each algorithm is examined at various phases of processing input for three individuals with sequence coverage of 10X and a reference sequence size of 4.68 Mb. The serial and parallel 1 algorithms perform each algorithm phase in the same order, therefore both are plotted in Figure 4.7. Data points were collected (in MB) using the Activity Monitor application included with Mac OS X at the following algorithm phases: after the initial parsing of the SNP input data, after the start of each parallel thread (one per input alignment file), after each alignment file has been processed, after all threads have been cleared (all parallel processes have completed), and when the program has finished (after writing of the output and before the program exits). The data points for the start of each thread and the thread clearing do not appear in the series for the serial algorithm as it does not utilize multiple threads.

As the execution order of algorithm phases differs for the parallel 2 algorithm, its data was plotted separately in Figure 4.8. The data points plotted are mostly the same, however, in Figure 4.8 there is a data point for the start of the algorithm, no data point for once the threads have cleared, and the data point for the SNP processing phase is collected after processing the alignment data. The start data point is used to

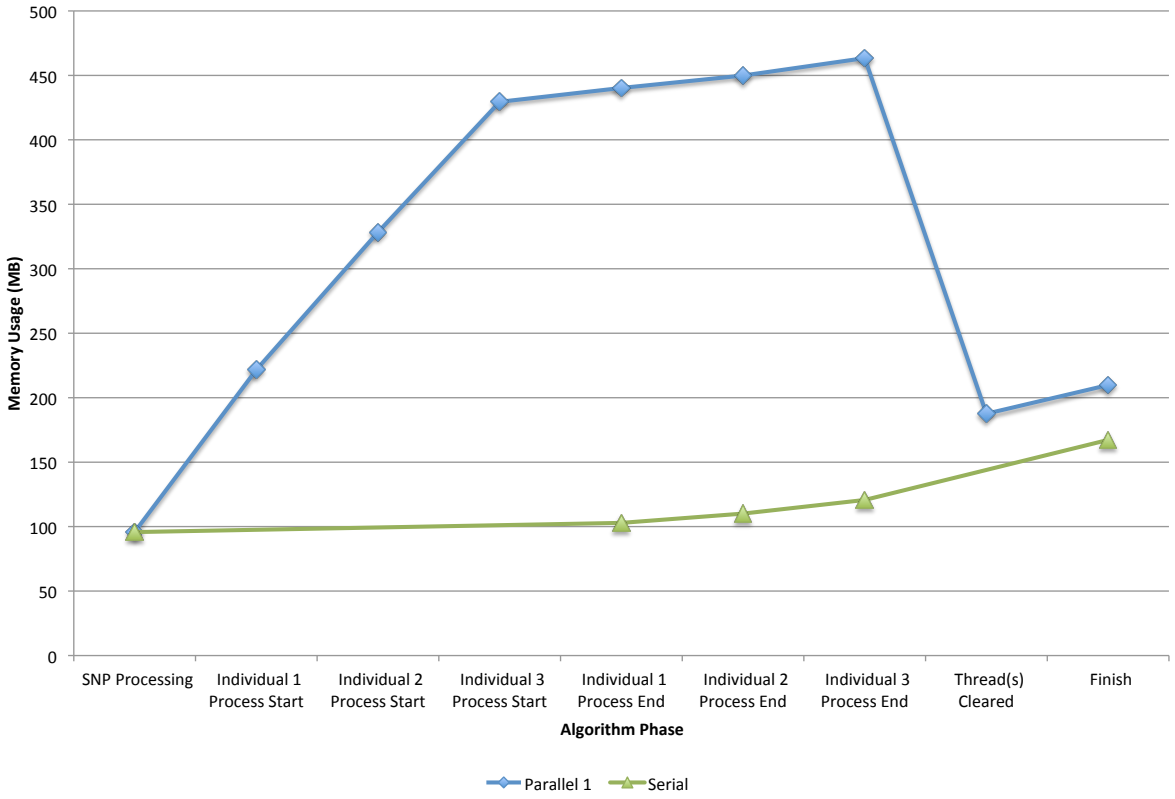

Figure 4.7: Chart of average memory usage during various phases of the serial and parallel 1 algorithms based on an input of three individuals. There are no data points for the serial algorithm for any of the parallelization specific algorithm phases. Each point in the series is a result of averaging three runs of the algorithm; the raw and average data are provided in Table 4.12.

show that very little memory is used to start each thread in the parallel 2 algorithm and there is no data point shown for when the threads have cleared as this value is always the same as the memory usage at the end of processing the final individual.

These figures indicate several important aspects of the algorithms: the serial algorithm, as expected, shows very slow memory growth; processing of alignment data from each individual results in an increase in memory usage; and total memory usage is highest at the end of the program. The slow growth of the serial algorithm can be attributed to the storage of alignment data for only the positions in the reference sequences that have a SNP in at least one line. The parallel 1 algorithm stores alignment data in the same way, but uses a large amount of memory to start each thread, which is then mostly released once processing the alignment data has finished and the thread can be cleared. This large increase in memory usage for starting each parallel thread can be attributed to the way Perl creates new threads; in Perl all data structures and

Algorithm	SNP Processing	Individual 1 Process Start	Individual 2 Process Start	Individual 3 Process Start	Individual 1 Process End	Individual 2 Process End	Individual 3 Process End	Thread(s) Cleared	Finish
Serial - Run 1	95.7				102.9	110.1	120.7		166.5
Serial - Run 2	95.9				102.9	110.2	120.7		167.6
Serial - Run 3	95.6				102.8	110.1	120.6		167.7
Parallel 1 - Run 1	95.8	224.7	329.1	421	434.8	446.9	462.1	190.9	208.9
Parallel 1 - Run 2	95.8	219.1	327.4	439.1	447.9	454.1	461.2	196.1	219.3
Parallel 1 - Run 3	95.8	221.1	328.6	428.4	438	448.6	466.8	175.6	201.3

(a)

Algorithm	SNP Processing	Individual 1 Process Start	Individual 2 Process Start	Individual 3 Process Start	Individual 1 Process End	Individual 2 Process End	Individual 3 Process End	Thread(s) Cleared	Finish
Serial	95.73				102.87	110.13	120.67		167.27
Parallel 1	95.8	221.63	328.37	429.5	440.23	449.87	463.37	187.53	209.83

(b)

Table 4.12: Memory usage results for various algorithm phases of the serial and parallel 1 algorithms. Each test was performed in triplicate (Runs 1-3) and the memory usage at each algorithm phase determined using the Activity Monitor application in Mac OS X (a). The results for each algorithm phase were then averaged (b) and charted in Figure 4.7.

variables created before the generation of a thread are copied entirely to the new thread. Creating the SNP data structure before generating threads to parse the alignment data allows the parallel 1 algorithm to only store alignment data for known SNP positions, similar to the serial algorithm. However, this structure is then copied to each new thread resulting in a spike in peak memory usage. Although this duplicated data is then released, resulting in a decrease in memory usage, peak memory usage is what limits a programs ability to run on specific hardware.

The parallel 2 algorithm moves the generation of the SNP data structure until after the alignment data has been processed. By not creating any data structures before generating new threads, parallel algorithm 2 solves the issue (duplication of data to all threads) of the parallel 1 algorithm. However, this means that alignment data must be stored for any reference position with a read aligned to it. The implementation of the parallel 2 algorithm results in a set of bit vectors (one per alignment element [A,T,C,G,*, and total]) for each individual, with each bit vector having a size that is 16 bits multiplied by the total reference size.

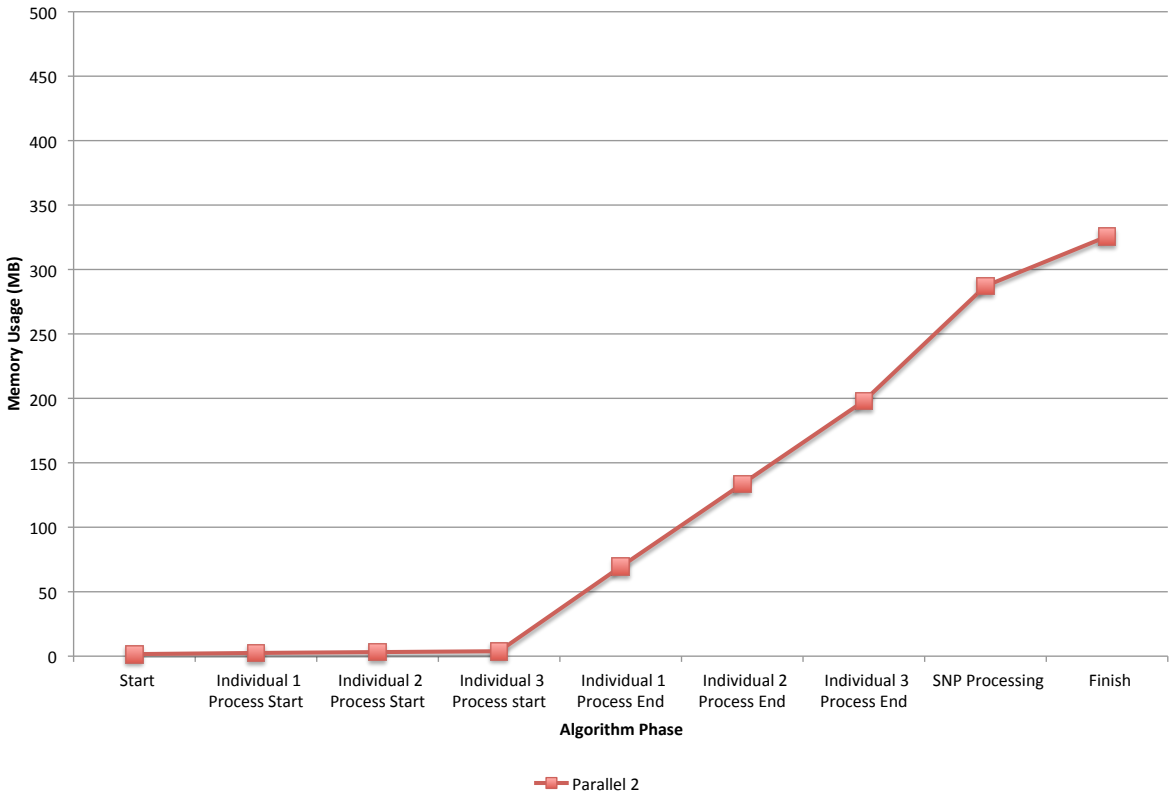

Figure 4.8: Chart of average memory usage during various phases of the parallel 2 algorithm based on an input of three individuals. Each point in the series is a result of averaging three runs of the algorithm; the raw and average data are provided in Table 4.13.

To evaluate the memory usage of each of the algorithms under a wider range of conditions, the test cases presented in Table 4.6 were again applied, but this time recording peak memory usage for each algorithm. To compare the algorithms, two linear regressions are performed, with one plotting memory versus unique SNP positions (V, Equation 4.2) multiplied by the number of individuals (n) (Figure 4.9) and the other plotting memory versus reference size (b) multiplied by the number of individuals (n) (Figure 4.10). These regressions were chosen based on the implementations of the algorithms and the analysis of the computational space complexity (Section 4.7.2). The serial and parallel 1 algorithms are expected to be linear in memory usage with respect to $V \cdot n$ and the parallel 2 algorithm is expected to be linear with respect to $b \cdot n$.

The regression of memory usage versus $V \cdot n$ shows that both the serial and parallel 1 algorithms have data points that are very well suited to the linear regression model (R^2 values of 0.99273 and 0.99388 respectively) indicating that these algorithms have memory usage patterns that are linear based on the SNP data input. This result matches the complexity analysis of Section 4.7.2 while providing actual memory

Algorithm	Start	Individual 1 Process Start	Individual 2 Process Start	Individual 3 Process start	Individual 1 Process End	Individual 2 Process End	Individual 3 Process End	SNP Processing	Finish
Parallel 2 - Run 1	1.6	2.4	3.2	3.8	69.3	133.9	198.3	287.6	326.2
Parallel 2 - Run 2	1.6	2.5	3.2	3.8	69.2	133.8	198.4	288	325.4
Parallel 2 - Run 3	1.6	2.5	3.1	3.8	69.7	133.4	197.1	285.9	325.1

(a)

Algorithm	Start	Individual 1 Process Start	Individual 2 Process Start	Individual 3 Process start	Individual 1 Process End	Individual 2 Process End	Individual 3 Process End	SNP Processing	Finish
Parallel 2	1.6	2.47	3.17	3.8	69.4	133.7	197.93	287.17	325.57

(b)

Table 4.13: Memory usage results for various algorithm phases of the parallel 2 algorithm. Each test was performed in triplicate (Runs 1-3) and the memory usage at each algorithm phase determined using the Activity Monitor application in Mac OS X (a). The results for each algorithm phase were then averaged (b) and charted in Figure 4.8.

estimates based on varying input sizes. The parallel 2 algorithm data points have a R^2 value of 0.94393 indicating a reasonable fit of the data points to the linear model. This is expected as the space complexity of this algorithm does have a component that depends on the number of SNPs and the number of individuals.

The regression of memory usage versus $b \cdot n$ shows that the parallel 2 algorithm is very well suited to the linear regression model (R^2 value of 0.99745) indicating that this algorithm has a memory usage pattern that is linear in the size of the reference and the number of individuals in the input. Further, the higher R^2 in this regression indicates that memory usage of the parallel 2 algorithm is more dependent on reference size than on the number of unique SNP positions. The serial algorithm has a R^2 value of 0.96313 and appears to be linear with respect to reference size and the number of individuals. We expect this result as both the size of reference sequence set and the number of individuals can affect the number of SNP positions that can be discovered and thus stored by our algorithms. Although the R^2 value of the parallel 1 algorithm indicates close proximity of the points to the regression model it seems possible from Figure 4.10 that its memory usage is non-linear with respect to reference size times the number of individuals. This is due to increases in the number of individuals resulting in an increase in the size of the SNP data structure as well as its duplication across threads.

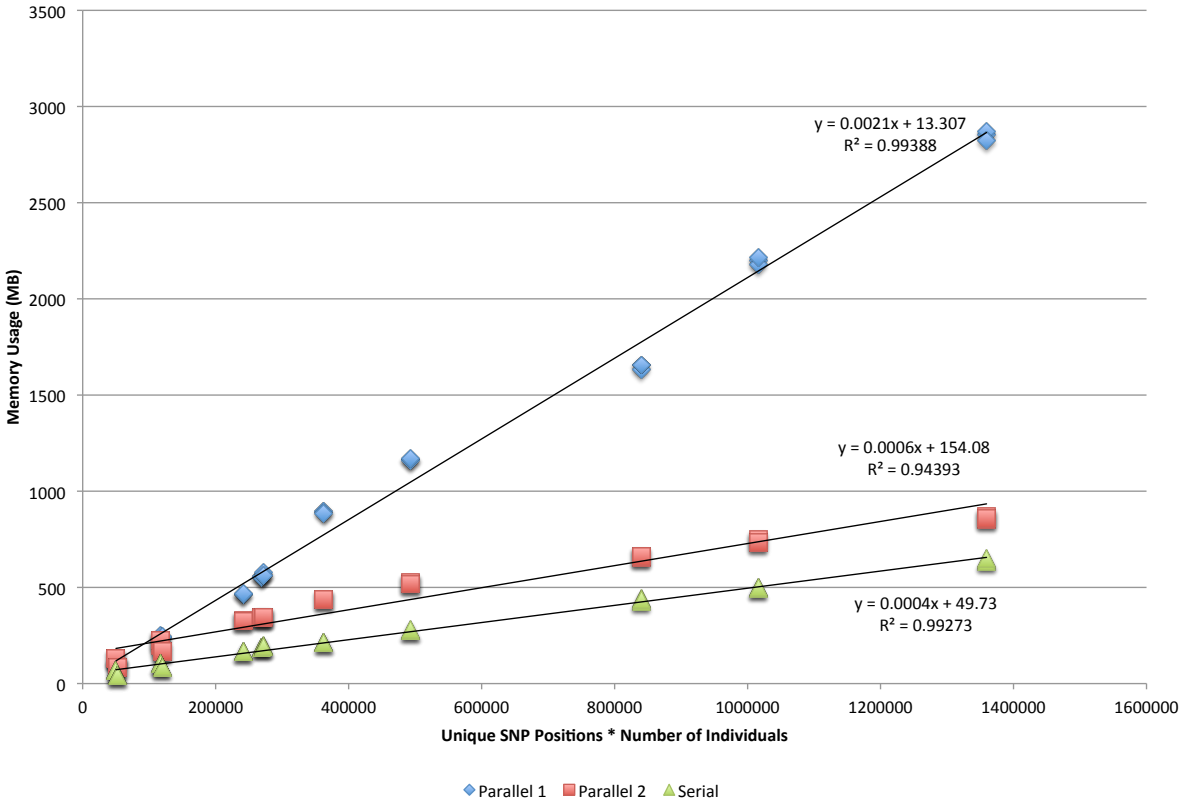

Figure 4.9: Results of linear regression analysis of memory versus unique SNP positions multiplied by the number of individuals for the serial, parallel 1, and parallel 2 algorithms. Data points are based on the number of unique SNP positions and individuals for each of the test cases outlined in Table 4.6. Each test case was performed in triplicate for each algorithm (Tables 4.14, 4.15, and 4.16 for the serial, parallel 1, and parallel 2 algorithms, respectfully).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Individuals * SNP Count	Peak Mem (MB)
Serial	1	4.68	10X	50044	67.2
Serial	1	4.68	10X	50044	67.5
Serial	1	4.68	10X	50044	67.7
Serial	2	4.68	10X	116812	103.2
Serial	2	4.68	10X	116812	103.4
Serial	2	4.68	10X	116812	102.8
Serial	3	4.68	10X	241506	166.5
Serial	3	4.68	10X	241506	167.6
Serial	3	4.68	10X	241506	167.7
Serial	4	4.68	10X	362392	214.8
Serial	4	4.68	10X	362392	215
Serial	4	4.68	10X	362392	213.9
Serial	5	4.68	10X	493000	273.4
Serial	5	4.68	10X	493000	275.7
Serial	5	4.68	10X	493000	282.1
Serial	6	4.68	10X	840150	428.6
Serial	6	4.68	10X	840150	442.1
Serial	6	4.68	10X	840150	435.6
Serial	7	4.68	10X	1015959	498.3
Serial	7	4.68	10X	1015959	491.8
Serial	7	4.68	10X	1015959	500.8
Serial	8	4.68	10X	1359360	643.8
Serial	8	4.68	10X	1359360	636.3
Serial	8	4.68	10X	1359360	649.4
Serial	3	2.34	10X	119685	82.7
Serial	3	2.34	10X	119685	82.3
Serial	3	2.34	10X	119685	86.6
Serial	3	1.17	10X	52272	41
Serial	3	1.17	10X	52272	42.9
Serial	3	1.17	10X	52272	40.6
Serial	3	4.68	50X	268794	186.8
Serial	3	4.68	50X	268794	189.7
Serial	3	4.68	50X	268794	187.6
Serial	3	4.68	100X	272211	195.7
Serial	3	4.68	100X	272211	193.5
Serial	3	4.68	100X	272211	193.8

Table 4.14: Memory usage results for changes in the number of unique SNPs multiplied by the number of individuals, measured using the Mac OS X variant of the *unix top* command, for each of three replicates of the test cases described in Table 4.6 for the serial algorithm (Section 4.5).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Individuals * SNP Count	Peak Mem (MB)
Parallel 1	1	4.68	10X	50044	109.4
Parallel 1	1	4.68	10X	50044	106.4
Parallel 1	1	4.68	10X	50044	106.3
Parallel 1	2	4.68	10X	116812	241.7
Parallel 1	2	4.68	10X	116812	243.7
Parallel 1	2	4.68	10X	116812	248.6
Parallel 1	3	4.68	10X	241506	462.1
Parallel 1	3	4.68	10X	241506	461.2
Parallel 1	3	4.68	10X	241506	466.8
Parallel 1	4	4.68	10X	362392	896.1
Parallel 1	4	4.68	10X	362392	896.1
Parallel 1	4	4.68	10X	362392	883.6
Parallel 1	5	4.68	10X	493000	1167.1
Parallel 1	5	4.68	10X	493000	1160.7
Parallel 1	5	4.68	10X	493000	1170.5
Parallel 1	6	4.68	10X	840150	1634.2
Parallel 1	6	4.68	10X	840150	1656.7
Parallel 1	6	4.68	10X	840150	1656.7
Parallel 1	7	4.68	10X	1015959	2199.5
Parallel 1	7	4.68	10X	1015959	2178.5
Parallel 1	7	4.68	10X	1015959	2214.7
Parallel 1	8	4.68	10X	1359360	2856.3
Parallel 1	8	4.68	10X	1359360	2870.6
Parallel 1	8	4.68	10X	1359360	2823.5
Parallel 1	3	2.34	10X	119685	238.2
Parallel 1	3	2.34	10X	119685	239.7
Parallel 1	3	2.34	10X	119685	243
Parallel 1	3	1.17	10X	52272	115.9
Parallel 1	3	1.17	10X	52272	121.1
Parallel 1	3	1.17	10X	52272	115.4
Parallel 1	3	4.68	50X	268794	555.8
Parallel 1	3	4.68	50X	268794	552.3
Parallel 1	3	4.68	50X	268794	558.4
Parallel 1	3	4.68	100X	272211	563.6
Parallel 1	3	4.68	100X	272211	575.3
Parallel 1	3	4.68	100X	272211	563

Table 4.15: Memory usage results for changes in the number of unique SNPs multiplied by the number of individuals, measured using the Mac OS X variant of the *unix top* command, for each of three replicates of the test cases described in Table 4.6 for the parallel 1 algorithm (Section 4.6.2).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Individuals * SNP Count	Peak Mem (MB)
Parallel 2	1	4.68	10X	50044	130.8
Parallel 2	1	4.68	10X	50044	130.3
Parallel 2	1	4.68	10X	50044	130
Parallel 2	2	4.68	10X	116812	224.7
Parallel 2	2	4.68	10X	116812	197.8
Parallel 2	2	4.68	10X	116812	225.8
Parallel 2	3	4.68	10X	241506	326.2
Parallel 2	3	4.68	10X	241506	325.4
Parallel 2	3	4.68	10X	241506	325.1
Parallel 2	4	4.68	10X	362392	436.5
Parallel 2	4	4.68	10X	362392	434.6
Parallel 2	4	4.68	10X	362392	439.1
Parallel 2	5	4.68	10X	493000	519
Parallel 2	5	4.68	10X	493000	523.2
Parallel 2	5	4.68	10X	493000	517.7
Parallel 2	6	4.68	10X	840150	662
Parallel 2	6	4.68	10X	840150	656.9
Parallel 2	6	4.68	10X	840150	659.1
Parallel 2	7	4.68	10X	1015959	735.3
Parallel 2	7	4.68	10X	1015959	750.8
Parallel 2	7	4.68	10X	1015959	735
Parallel 2	8	4.68	10X	1359360	850.5
Parallel 2	8	4.68	10X	1359360	867
Parallel 2	8	4.68	10X	1359360	859.1
Parallel 2	3	2.34	10X	119685	167.9
Parallel 2	3	2.34	10X	119685	168.3
Parallel 2	3	2.34	10X	119685	167.9
Parallel 2	3	1.17	10X	52272	84.8
Parallel 2	3	1.17	10X	52272	84.2
Parallel 2	3	1.17	10X	52272	84.9
Parallel 2	3	4.68	50X	268794	343.1
Parallel 2	3	4.68	50X	268794	343.5
Parallel 2	3	4.68	50X	268794	344.7
Parallel 2	3	4.68	100X	272211	343.6
Parallel 2	3	4.68	100X	272211	346.4
Parallel 2	3	4.68	100X	272211	346

Table 4.16: Memory usage results for changes in the number of unique SNPs multiplied by the number of individuals, measured using the Mac OS X variant of the *unix top* command, for each of three replicates of the test cases described in Table 4.6 for the parallel 2 algorithm (Section 4.6.3).

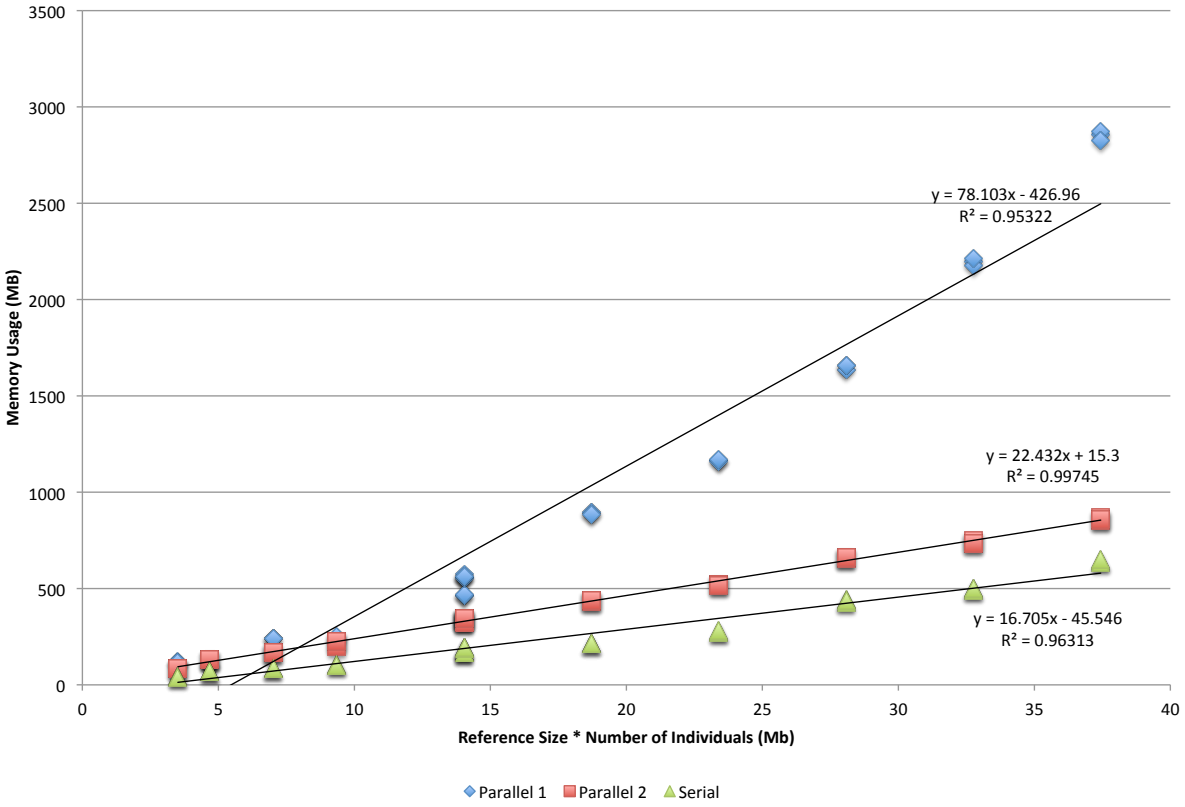

Figure 4.10: Results of linear regression analysis of memory versus reference size multiplied by the number of individuals for the serial, parallel 1, and parallel 2 algorithms. Data points are based on the number of unique SNP positions and individuals for each of the test cases outlined in Table 4.6. Each test case was performed in triplicate for each algorithm (Tables 4.17, 4.18, and 4.19 for the serial, parallel 1, and parallel 2 algorithms, respectfully).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Individuals * Reference Size	Peak Mem
Serial	1	4.68	10X	4.68	67.2
Serial	1	4.68	10X	4.68	67.5
Serial	1	4.68	10X	4.68	67.7
Serial	2	4.68	10X	9.36	103.2
Serial	2	4.68	10X	9.36	103.4
Serial	2	4.68	10X	9.36	102.8
Serial	3	4.68	10X	14.04	166.5
Serial	3	4.68	10X	14.04	167.6
Serial	3	4.68	10X	14.04	167.7
Serial	4	4.68	10X	18.72	214.8
Serial	4	4.68	10X	18.72	215
Serial	4	4.68	10X	18.72	213.9
Serial	5	4.68	10X	23.4	273.4
Serial	5	4.68	10X	23.4	275.7
Serial	5	4.68	10X	23.4	282.1
Serial	6	4.68	10X	28.08	428.6
Serial	6	4.68	10X	28.08	442.1
Serial	6	4.68	10X	28.08	435.6
Serial	7	4.68	10X	32.76	498.3
Serial	7	4.68	10X	32.76	491.8
Serial	7	4.68	10X	32.76	500.8
Serial	8	4.68	10X	37.44	643.8
Serial	8	4.68	10X	37.44	636.3
Serial	8	4.68	10X	37.44	649.4
Serial	3	2.34	10X	7.02	82.7
Serial	3	2.34	10X	7.02	82.3
Serial	3	2.34	10X	7.02	86.6
Serial	3	1.17	10X	3.51	41
Serial	3	1.17	10X	3.51	42.9
Serial	3	1.17	10X	3.51	40.6
Serial	3	4.68	50X	14.04	186.8
Serial	3	4.68	50X	14.04	189.7
Serial	3	4.68	50X	14.04	187.6
Serial	3	4.68	100X	14.04	195.7
Serial	3	4.68	100X	14.04	193.5
Serial	3	4.68	100X	14.04	193.8

Table 4.17: Memory usage results for changes in the reference size (Mb) multiplied by the number of individuals, measured using the Mac OS X variant of the *unix top* command, for each of three replicates of the test cases described in Table 4.6 for the serial algorithm (Section 4.5).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Individuals * Reference Size	Peak Mem
Parallel 1	1	4.68	10X	4.68	109.4
Parallel 1	1	4.68	10X	4.68	106.4
Parallel 1	1	4.68	10X	4.68	106.3
Parallel 1	2	4.68	10X	9.36	241.7
Parallel 1	2	4.68	10X	9.36	243.7
Parallel 1	2	4.68	10X	9.36	248.6
Parallel 1	3	4.68	10X	14.04	462.1
Parallel 1	3	4.68	10X	14.04	461.2
Parallel 1	3	4.68	10X	14.04	466.8
Parallel 1	4	4.68	10X	18.72	896.1
Parallel 1	4	4.68	10X	18.72	896.1
Parallel 1	4	4.68	10X	18.72	883.6
Parallel 1	5	4.68	10X	23.4	1167.1
Parallel 1	5	4.68	10X	23.4	1160.7
Parallel 1	5	4.68	10X	23.4	1170.5
Parallel 1	6	4.68	10X	28.08	1634.2
Parallel 1	6	4.68	10X	28.08	1656.7
Parallel 1	6	4.68	10X	28.08	1656.7
Parallel 1	7	4.68	10X	32.76	2199.5
Parallel 1	7	4.68	10X	32.76	2178.5
Parallel 1	7	4.68	10X	32.76	2214.7
Parallel 1	8	4.68	10X	37.44	2856.3
Parallel 1	8	4.68	10X	37.44	2870.6
Parallel 1	8	4.68	10X	37.44	2823.5
Parallel 1	3	2.34	10X	7.02	238.2
Parallel 1	3	2.34	10X	7.02	239.7
Parallel 1	3	2.34	10X	7.02	243
Parallel 1	3	1.17	10X	3.51	115.9
Parallel 1	3	1.17	10X	3.51	121.1
Parallel 1	3	1.17	10X	3.51	115.4
Parallel 1	3	4.68	50X	14.04	555.8
Parallel 1	3	4.68	50X	14.04	552.3
Parallel 1	3	4.68	50X	14.04	558.4
Parallel 1	3	4.68	100X	14.04	563.6
Parallel 1	3	4.68	100X	14.04	575.3
Parallel 1	3	4.68	100X	14.04	563

Table 4.18: Memory usage results for changes in the reference size (Mb) multiplied by the number of individuals, measured using the Mac OS X variant of the *unix top* command, for each of three replicates of the test cases described in Table 4.6 for the parallel 1 algorithm (Section 4.6.2).

Algorithm	Individuals	Reference Size (Mb)	Coverage	Individuals * Reference Size	Peak Mem
Parallel 2	1	4.68	10X	4.68	130.8
Parallel 2	1	4.68	10X	4.68	130.3
Parallel 2	1	4.68	10X	4.68	130
Parallel 2	2	4.68	10X	9.36	224.7
Parallel 2	2	4.68	10X	9.36	197.8
Parallel 2	2	4.68	10X	9.36	225.8
Parallel 2	3	4.68	10X	14.04	326.2
Parallel 2	3	4.68	10X	14.04	325.4
Parallel 2	3	4.68	10X	14.04	325.1
Parallel 2	4	4.68	10X	18.72	436.5
Parallel 2	4	4.68	10X	18.72	434.6
Parallel 2	4	4.68	10X	18.72	439.1
Parallel 2	5	4.68	10X	23.4	519
Parallel 2	5	4.68	10X	23.4	523.2
Parallel 2	5	4.68	10X	23.4	517.7
Parallel 2	6	4.68	10X	28.08	662
Parallel 2	6	4.68	10X	28.08	656.9
Parallel 2	6	4.68	10X	28.08	659.1
Parallel 2	7	4.68	10X	32.76	735.3
Parallel 2	7	4.68	10X	32.76	750.8
Parallel 2	7	4.68	10X	32.76	735
Parallel 2	8	4.68	10X	37.44	850.5
Parallel 2	8	4.68	10X	37.44	867
Parallel 2	8	4.68	10X	37.44	859.1
Parallel 2	3	2.34	10X	7.02	167.9
Parallel 2	3	2.34	10X	7.02	168.3
Parallel 2	3	2.34	10X	7.02	167.9
Parallel 2	3	1.17	10X	3.51	84.8
Parallel 2	3	1.17	10X	3.51	84.2
Parallel 2	3	1.17	10X	3.51	84.9
Parallel 2	3	4.68	50X	14.04	343.1
Parallel 2	3	4.68	50X	14.04	343.5
Parallel 2	3	4.68	50X	14.04	344.7
Parallel 2	3	4.68	100X	14.04	343.6
Parallel 2	3	4.68	100X	14.04	346.4
Parallel 2	3	4.68	100X	14.04	346

Table 4.19: Memory usage results for changes in the reference size (Mb) multiplied by the number of individuals, measured using the Mac OS X variant of the *unix top* command, for each of three replicates of the test cases described in Table 4.6 for the parallel 2 algorithm (Section 4.6.3).

4.10 Parallelization Bottlenecks

An important aspect of evaluating the performance of parallel algorithms is to try and determine what computational resources may be limiting performance. The two most common bottlenecks are access to storage, such as the computer's hard disk or memory, and CPU time. The most common approach to determining if a parallel program is CPU or memory bound is to evaluate the source code of the program and estimate the number of CPU and memory actions. The number of CPU actions are divided by the number of instructions the CPU can perform per clock cycle multiplied by the clock speed of the CPU to give the CPU bound. The number of memory operations are divided by the memory access speed to give the memory bound. This type of analysis is often performed on algorithms with very clear or easily simplified CPU and memory operations [54]. Additionally, in some programming languages, the estimated instruction sets can be generated using the compiler or other software [75]. However, no such utilities exist for profiling CPU and memory instructions in the Perl language. Combined with the complexity of the three algorithms presented here it was beyond the scope of this thesis to produce an accurate representation of the CPU and memory boundaries using this method.

The performance limits of the parallel algorithms are not evaluated by inspecting their code directly, instead the approach taken is to determine these limits by observing changes in the running time. By fixing the size of the input to eight individuals and varying the number of available CPU threads (amount of possible parallelization) observations can be made which provide insight into how the algorithms are limited.

Figure 4.11 shows the results of plotting the running time versus the number of CPU threads for our parallel algorithms. There is an obvious decrease in running time as the number of CPU threads increased from one to four, while the pattern for five, six and seven threads appears to indicate the levelling off (and in some cases increasing) of running time. An explanation for the levelling off seen in the trials of 4, 5, 6, or 7 threads is that the upper bound of the time required for processing the alignment data can be estimated using the equation $\lceil n/t \rceil$ (Equation 4.6), where n is the number of individuals and t the number of threads available. This function results in the same upper bound (2) for eight individuals when the number of threads is 4, 5, 6, or 7 and as such the running time of these trials should be the same. Variation in the running times for 4, 5, 6, or 7 threads may be due to other demands, such as those from background processes, on the test system and might decrease if the results from several trials were averaged. When the number of threads is increased to eight, decreases in running time of 9% and 11%, for the parallel 1 and parallel 2 algorithms respectively, are observed versus the next fastest running time. This decrease in running time is expected,

as increasing the number of CPU threads from seven to eight allows all of the individuals to be started at the same time (upper bound of 1).

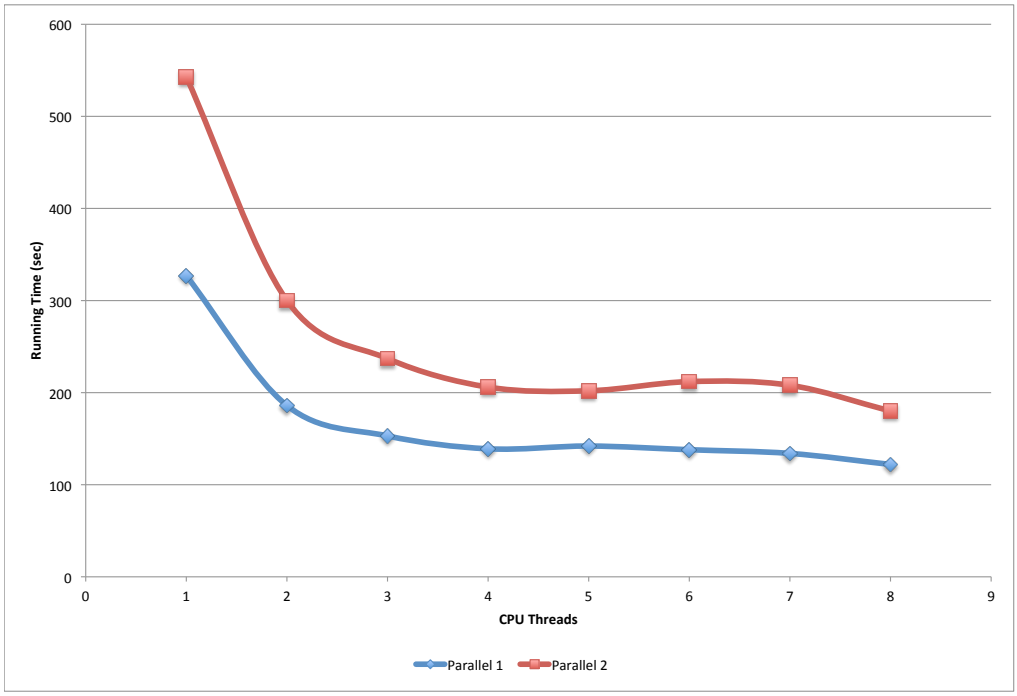

Figure 4.11: Running time results for input of 8 individuals, each with 10X coverage, aligned to a 4.68 Mb reference sequence set allowing parallel 1 and parallel 2 algorithms access to different numbers of threads. Raw series data is presented in Table 4.20.

Based on these observations, one potential cause of the limited speedup of the parallel algorithms seen in Section 4.9.1 is the choice of programming language used in the implementation. This is suggested by the knowledge that each thread in the parallel algorithms acts independently on its input. This means that the algorithms themselves are not limiting the parallelization performance and that the theoretical maximum speedup values calculated using Amdahl's law should be obtainable. Future work on further investigating parallelization bottlenecks as well as improvements to the parallelization will be discussed in Chapter 5.

Number of threads	Running Time (sec)	
	Parallel 1	Parallel 2
1	327	543
2	186	300
3	153	237
4	139	206
5	142	202
6	138	212
7	134	208
8	122	180

Table 4.20: Running time results for the parallel 1 and parallel 2 algorithms when given access to 1-8 parallel threads. Input data was for 8 individuals, each with 10X coverage, aligned to a 4.68 Mb reference sequence set.

4.11 Algorithm Selection

Assuming that the goal is to minimize run time, selection of an algorithm for combining SNP data into the multisample output is dependent on available memory. As shown in Figures 4.9 & 4.10, memory usage of the algorithms is due to several factors including the number of input individuals, the size of the reference set, and the diversity among the individuals (number of unique SNP positions). As the selection of an algorithm is also highly dependent on the amount of available memory, an approach for estimating the memory usage of each algorithm is discussed. Estimated memory requirements can then be compared to available memory to determine if a particular algorithm is suitable to an application.

The parallel 1 algorithm has the fastest running time (Figure 4.6) and therefore should be checked for suitability first. For the Perl implementation, its memory usage is dependent on the number of individuals and the number of unique SNP positions. As it has the fastest memory usage growth rate (Figure 4.9), it is suitable for applications with a low number of SNP positions and/or a low number of individuals. As the memory for each thread is not released until all threads have completed, this approach is generally not suitable for very large numbers of individuals. Determining the number of SNP positions can be done by extracting the reference name and position of each SNP from every individual SNP report, then sorting them and removing duplicates. This number can then be multiplied by the number of individuals and substituted

for x in the slope ($y = 0.0021x + 13.307$) of the parallel 1 algorithm's linear model (Figure 4.9) to estimate the required memory usage.

The parallel 2 algorithm in Perl has a memory usage which is dependent on the number of individuals and the size of the reference sequence set. This algorithm is suitable for applications where the reference sequence size is moderate and the number of individuals is low. Each individual has six bit vectors (one per alignment element), each of which store at most the length of the reference sequence set multiplied by 16 (the number of bits per reference position). Determining the size of the reference is quite simple, we simply add up the length of each individual reference sequence in the set. This number can then be multiplied by the number of individuals and substituted for x in the slope ($y = 22.432x + 15.3$) of the parallel 2 algorithm's linear model (Figure 4.10) to estimate the required memory usage.

When neither the parallel 1 nor parallel 2 options are immediately suitable due to excess memory usage there are two options. The first is to evaluate the serial algorithm to determine if it will allow for the processing of the data within the required memory space and with an acceptable running time. Since the serial algorithm is dependent on the number of individuals and the number of unique SNP positions, a similar estimation to the parallel 1 algorithm can be performed. Once the number of SNP positions have been determined it can be multiplied by the number of individuals and substituted for x in the slope ($y = 0.0004x + 49.73$) of the serial algorithm's linear model (Figure 4.9) to estimate the required memory usage. Running time can be estimated by summing the number of aligned reads across all SAM files in the input and then putting the sum into the slope ($y = 0.00005x + 13.986$) of the serial algorithm's linear model for running time. If either the predicted memory usage or running time make the computation of the multisample SNP table impossible, then the input may be split to allow for the use of one of the parallel algorithms. Splitting of the input data should be done by reference sequence position. This allows the algorithm to still provide context for a SNP across all of the input individuals with no change in the algorithm results and minimal overhead when combining the output of split data.

4.12 Filtering Of SNP results

For the purpose of designing SNP genotyping arrays we need to select SNPs that will result in robust markers. Ideally, discovery of SNPs would be both comprehensive and 100% accurate, allowing us to quickly assess the SNP for suitability. However, due to biological complexities and errors introduced during the sequencing process this is not the case. In reality, SNP discovery software can vary significantly in called SNP sets

using the same data sources [85], illustrating the complexity of SNP discovery. SNP discovery is particularly complicated in larger more complex genomes, where polyploidy, repetitive elements, and duplication of genomic regions are more common. Our approach, in dealing with non-perfect SNP data, is to combine evidence from multiple individuals and screen on multiple criteria to select, from a large pool of available SNPs, a subset that results in robust markers. As no available software provided the filtering we required, we developed our own filtering methods as part of the design of several Illumina, Inc. GoldenGate and Infinium SNP genotyping arrays.

4.12.1 Filtering Raw SNP data

This section describes the implementation of a filtering method which utilizes the robust output format generated by the algorithms described in Section 4.4. This output has the SNP id, reference id and position, flanking sequence if available, the reference allele, and for each individual surveyed, the SNP call, depth and frequency data (Figure 4.2). This comprehensive data allows for the filtering of SNPs based on the following criteria:

1. the frequency of individuals with null calls,
2. the frequency of individuals with heterozygous calls,
3. the frequency of individuals with the reference allele,
4. confidence in the SNP call,
5. appropriate flanking sequence data (application specific).

It would be desirable to combine these criteria in an optimal way by systematically varying their influence on inclusion in the filtered data set, and then by measuring their efficacy. But this is not, as yet, practical as it would require creating a SNP genotyping array for each combination, which is too costly. Thus a set of heuristics will be described and their success will be measured against other genotyping arrays that have been created. The method employed is as follows.

The frequency of null calls is easily calculated by looping over the calls and counting the number of individuals called as "X". The number of null calls is then divided by the total number of individuals and if that value is greater than a user defined cutoff, then the SNP is excluded. The frequency of heterozygous SNPs is similarly calculated and filtered by counting the number of individuals with more than one allele in their call field (i.e. A/T) and determining the frequency with respect to non null individuals. The frequency

of the reference allele is calculated by counting the number of individuals with the same base call as the reference base and dividing by the number of non null individuals. The value must fall into a user defined range (for example 0.25–0.75), otherwise the SNP is filtered out. This ensures that the reference alleles are not under or over represented in the individual samples and is more commonly recognized as the minor allele frequency. To calculate the confidence of a SNP call, individuals with high quality SNP calls (read depth above a user defined threshold and 100 percent SNP frequency) and individuals with marginal SNP calls (read depth lower than the threshold and 100 percent SNP frequency or read depth higher than the threshold and greater then 80 percent SNP frequency) are counted. To be a confident SNP, the number of high quality SNP calls has to be greater than the number of marginal SNP calls and the combined total must be above a user defined threshold when divided by the number of non null individuals. For the Illumina Infinium array designs, a flanking sequence of at least 60 bp on one side of the SNP is required. Further, the flanking sequence could not contain SNPs. For the Illumina GoldenGate array designs, a SNP free flanking sequence of 100 bp is required on both sides of the SNP. Since the flanking sequence in the output has flanking SNP positions converted to IUPAC ambiguity codes, SNPs are filtered if they do not have an appropriate length of flanking sequence without an ambiguity code.

4.13 Selection Of SNPs

In many cases the number of SNPs remaining after filtering (using the previously described approach) is larger than the number of SNPs to be included onto the SNP array. Therefore, a subset of the remaining SNPs are selected for inclusion onto the SNP array. The two methods used in this thesis are selection of SNPs based on their distribution in the reference sequence set (described in Section 4.13.1) and selection of SNPs based on the number of alignments of the Illumina probe sequence to the reference sequence set (described in Section 4.13.2).

4.13.1 SNP Selection Based On Distribution In Reference

The first method employed for SNP selection was to select SNPs based on their distribution throughout the reference sequences, in order to generate a relatively even distribution of SNPs across the reference genome. This is accomplished by first identifying the subset of reference sequences represented by SNPs in the filtered SNP set. Next, the number of SNPs to select from each reference sequence is determined by multiplying the total number of SNPs to be selected by the length of the reference sequence divided by the total length of

all reference sequences. This ensures that large reference sequences are allocated a greater numbers of SNPs, allowing better distribution of SNPs across the reference set. Each reference sequence is then binned into ranges of base pair positions, where the number of bins is determined by dividing the length of the sequence by the number of SNPs allocated to it. Bins for each reference are then checked to ensure at least one SNP is present in every bin. If a SNP has not been located in every bin, the reference sequence is re-binned using a larger number of bins. This process reduces bin size, until either all SNPs are allocated to a bin or a user defined threshold for minimum bin size is crossed. If more than one SNP is found in a bin then the algorithm selects the SNP closest to the middle of the bin, where the middle of the bin is the midpoint of the base pair range. If the total number of SNPs selected by the algorithm does not match the total number of SNPs that were to be selected, then the remaining SNPs are chosen at random.

4.13.2 SNP Selection Based On Illumina Probe Matches

As part of the Illumina procedure for developing genotyping arrays, the SNP id and flanking sequence of each candidate SNP is submitted to Illumina to undergo their probe design and evaluation. A probe is a subsequence of the SNPs flanking sequence that is attached to a bead on the array. In some species, probes from the submitted SNPs are tested against the available genome sequence by Illumina in order to determine if the probe comes from an ambiguous genomic position. However, as most non-model species do not have publicly available genome sequences, this check is not available.

In order to replicate this methodology, the probe sequences for all filtered SNPs were obtained from Illumina. These sequences are then matched to the reference sequences using the open source alignment tool BLAT [53]. These alignments are then parsed to determine the number of times the probe sequence from a particular SNP matched to the reference sequence set. SNPs are then ranked based on the number of times their probe sequence matches the reference sequence set and SNPs with fewer probe matches are preferentially selected.

4.14 Applications And Comparisons To Other Methods

The algorithms developed in Sections 4.4, 4.12, and 4.13 have been used for the development of Illumina GoldenGate and Infinium SNP genotyping arrays for both diploid (lentil) [99] and polyploid (camelina, Canola) non-model crop species [20, 101]. While this section will focus mainly on the use of these methods in polyploid species, it is worth mentioning that the Illumina GoldenGate SNP genotyping array with 1,536

SNPs developed for lentil enabled the first comprehensive genetic map in lentil to be produced [99].

Comparisons of the polyploid arrays developed using methods from this thesis will be made to three published plant genotyping arrays: a 90,000 SNP wheat array, a 7,867 SNP apple array, and a 8,303 SNP potato array. These arrays were selected based primarily on the similar genomic complexities of these species and the quality of the publication. Both the apple and potato arrays use a wide variety of SNP filters such as read depth, duplicate genes, multi-allelic SNPs, and Illumina ADT score. Further, both select SNPs based mainly on distribution across the respective genome [17, 37]. The wheat array uses two main filters for SNPs, removal of SNPs found in annotated repeat regions and removal of SNPs in close proximity to exon-intron junctions [120]. No details are provided on the selection of filtered SNPs to be used on the wheat array.

In camelina, an Illumina GoldenGate SNP array with 768 SNPs was developed. Of these 768 SNPs, 534 (69.5%) could be mapped using a single population. This compares favourably to other SNP arrays in polyploids such as wheat, where single population polymorphism ranged from 15.4% to 25.9% (43.7% across 8 populations) [120] and apple, where 72.2% of SNPs were polymorphic across 8 populations [17].

In Canola two arrays were developed, an Illumina Infinium SNP array containing 6,000 SNPs (known as the *Brassica napus* 6K array) and then a second 58,464 SNP Infinium array (known as the *Brassica napus* 60K array). Filtering of raw SNP output as discussed in Section 4.12 is important for the quality of the designed array. Both the 6K array and the 60K arrays were developed using reads sequenced from both Roche/454 and Illumina platforms. Due to the differences in the throughput and common error profiles of the two sequencing technologies (Section 2.7.2), filtering of SNPs based on read depth was performed at two different levels based on the sequencer type. Roche/454 reads were filtered using a lower read depth requirement as these sequencers produce fewer reads. However, since the common error type for this platform is insertions/deletions which were not the focus of this study, lower read depth did not appear to result in decreased confidence. Table 4.21 provides a detailed breakdown of the number of SNPs excluded at each filtering step as described in Section 4.12.1 for the 60K array.

SNPs for the 6K array were selected based on their distribution across the reference sequences (Section 4.13.1) while SNPs for the 60K array were selected using the probe uniqueness method described in Section 4.13.2 and subsequently the distribution across the reference sequences. Of the SNPs submitted to Illumina to be placed on the arrays, 5,506 (91.8%) of the SNPs from the 6K and 52,157 (89.2%) of the SNPs from the 60K passed the manufacturing process and were included on the respective arrays. This level of attrition between the submission and manufacturing phases is in line with other Illumina Infinium arrays [120], [29]. As these SNP arrays were developed in partnership with industry leaders, not all of the SNPs on the 6K and

Filter Method	SNPs excluded	SNP Count
None	0	24,528,374
Flanking Sequence	18,619,172	5,909,202
Multi-Allele SNP	7,671	5,901,531
Confidence	5,742,443	159,088
Illumina ADT Score (<0.6)	33,556	125,532
Transversions	1,318	124,214

Table 4.21: Breakdown of SNPs excluded and remaining at the each filtering stage described in Section 4.12.1 during the design of the Brassica 60K array. For the 60K array the flanking sequence length was 60 bp, SNPs with multiple alternate alleles were removed, a confidence threshold of 0.8 was used (80 percent of non-null lines must be either high or marginal confidence with the majority being high confidence). Additionally, SNPs with an Illumina design score (ADT score) less than 0.6 and transversions (A/C, A/T, G/C, G/T, C/A, T/A, C/G, or T/G) were also excluded.

60K arrays were developed using the processes described in this thesis. Of the 5,506 SNPs on the *Brassica napus* 6K array, 4,966 were designed using the methods from this thesis and of the 52,157 SNPs on the 60K array, 38,793 were designed using the methods from this thesis.

The 6K and 60K arrays were surveyed using a single population with 2,494 (50.2%) and 21,859 (56.3%) SNPs called as polymorphic in the 6K and 60K arrays, respectively. Initially, the level of polymorphic loci observed for the Brassica 6K array was used in comparisons to the highest published results for similar array analyses of the complex genomes of wheat and potato. Comparisons were made using a 2x2 contingency table and the Pearson's Chi Squared test, with the null hypothesis that there is no difference between the 6K array and the array to which it is being compared. The chi squared test was performed on the contingency table using the R programming language's built in *chisq.test* function. In wheat, where single population polymorphism results ranged from 15.4% to 25.9% [120], the chi squared test statistic was $\chi^2 = 1399.061$ and the p-value $< 2.2e-16$. The significance level of this p-value means that the null hypothesis is rejected, indicating that there is a significant difference between the 6K and wheat results in terms of the level of polymorphism. As higher levels of polymorphism are desired, it can be concluded that the approach to array design in the 6K (and the 60K by extension as the level of polymorphism is higher) is superior to that used for array design in wheat. In potato, where single population polymorphism ranged from 24.0% to 29.6% [29], the chi squared test statistic was $\chi^2 = 567.5242$ and the p-value $< 2.2e-16$. The p-value again indicates

that the design approach for the 6K and 60K SNP arrays is superior to that employed for the design of the potato array. The caveat being that these observed differences will also reflect the true level of biological variation between the parents of the mapping populations used in each of these studies.

Using the Illumina GenomeStudio software, which processes the SNP genotyping signal data for a panel of individuals at each SNP on the array and clusters the individuals by their genotypes, the genotype for every individual at every SNP on the array can be determined. Genotype clusters are generated by GenomeStudio using a normalized theta value, where theta ranges from zero to one and a theta of zero represents pure A genotype signal and a theta of one represents pure B genotype signal. Figure 4.12 shows an example of a clear three cluster SNP from the Brassica 60K array as represented in a graph. Each dot on the graph is the genotype of an individual screened on the array. If the individual falls within the darkly shaded area (and is the same colour) then the individual is part of that cluster. The generation of three clear genotype clusters from the array hybridization data is a measure of how well the array design method is able to target simple SNPs (Section 2.3.2) in complex polyploid genomes. Genotyping non-simple SNPs, such as hemi-SNPs, results in complex genotyping patterns complicating downstream analysis. Figure 4.13, shows a SNP from the Brassica 60K array which produces a complex genotyping pattern. This SNP results in 5 genotype clusters and is likely results from the SNP assay querying two homologous loci, each segregating hemi-SNPs. The GenomeStudio parameters used to call three cluster SNPs are as follows:

1. AA cluster theta mean of < 0.2 and BB cluster theta mean of > 0.8 – these values are used to identify SNPs with good separation of A and B alleles,

2. AB frequency of < 0.1 – used to eliminate SNPs which have a high number of heterozygotes indicating that the SNP is not a simple SNP,

3. AA and BB frequency is > 0 and minor allele frequency is > 0.01 – used to eliminate monomorphic SNPs.

To determine the number of clear three cluster SNPs present on the 6K array, a set of 399 diverse *Brassica napus* inbred individuals were tested, resulting in 929 (18.7%) clear three cluster SNPs as called by the GenomeStudio software. For the 60K SNP array, the number of tested individuals increased to 449, and resulted in 26,270 (67.7%) SNPs called as having three clear genotype clusters. The SNPs from the 60K array can be further divided based on the predicted uniqueness of their probes, as described in Section 4.13.2. SNPs with single match probes as well some SNPs with two probe matches were present on the array. The breakdown of the number of SNPs in each set, as well as the number of clear three cluster SNPs

is given in Table 4.22. The results of the 60K array clearly show an improvement in the percentage of clear three cluster SNP calls over that of the 6K array. To further validate this observation, a 2x2 contingency table and the chi squared test were used to evaluate if a statistically significant difference exists between the SNP selection methods; the null hypothesis of this test is that no difference exists between the methods. The results of the test were a $\chi^2 = 4495.777$ and a p-value $< 2.2e-16$. This p-value indicates that there is a statistically significant difference between the SNP selection methods. An additional chi squared test was performed to assess if a difference exists between SNPs with single probe matches and those with two probe matches, resulting in a $\chi^2 = 9238.348$ and a p-value $< 2.2e-16$ indicating that there is a significant difference between SNP types. The results of these statistical tests show the importance of careful selection of SNP markers. Expanding the analysis further, the number of clear three cluster probes in the *Brassica napus* 60K array were compared to those presented in the wheat 90K array [120]. The number of clear three cluster SNPs on the wheat array was 20,785 which represents (25.5%) of the 81,587 total SNPs on the wheat array. Again, the chi squared test was performed indicating a significant difference ($\chi^2 = 19704.74$, p-value $< 2.2e-16$) in the number of three cluster SNPs between the *Brassica napus* 60K array and the wheat 90K array. Other polyploid crops, such as apple—where more than 50% of SNP markers had clusters with no clear segregation pattern [115], have also struggled with the development of array markers which result in clear three cluster SNPs.

SNP Type	3-Cluster SNPs	Total SNPs	Percentage 3-Cluster
Single Probe Match	23,444	28,928	81.0%
Two Probe Matches	2,826	9,865	28.6%

Table 4.22: Breakdown of cluster types for the *Brassica napus* 60K array SNPs, designed using the methods developed in the thesis work, based on the number matches of the SNP probe to the reference sequences as determined using the SNP selection method from Section 4.13.2.

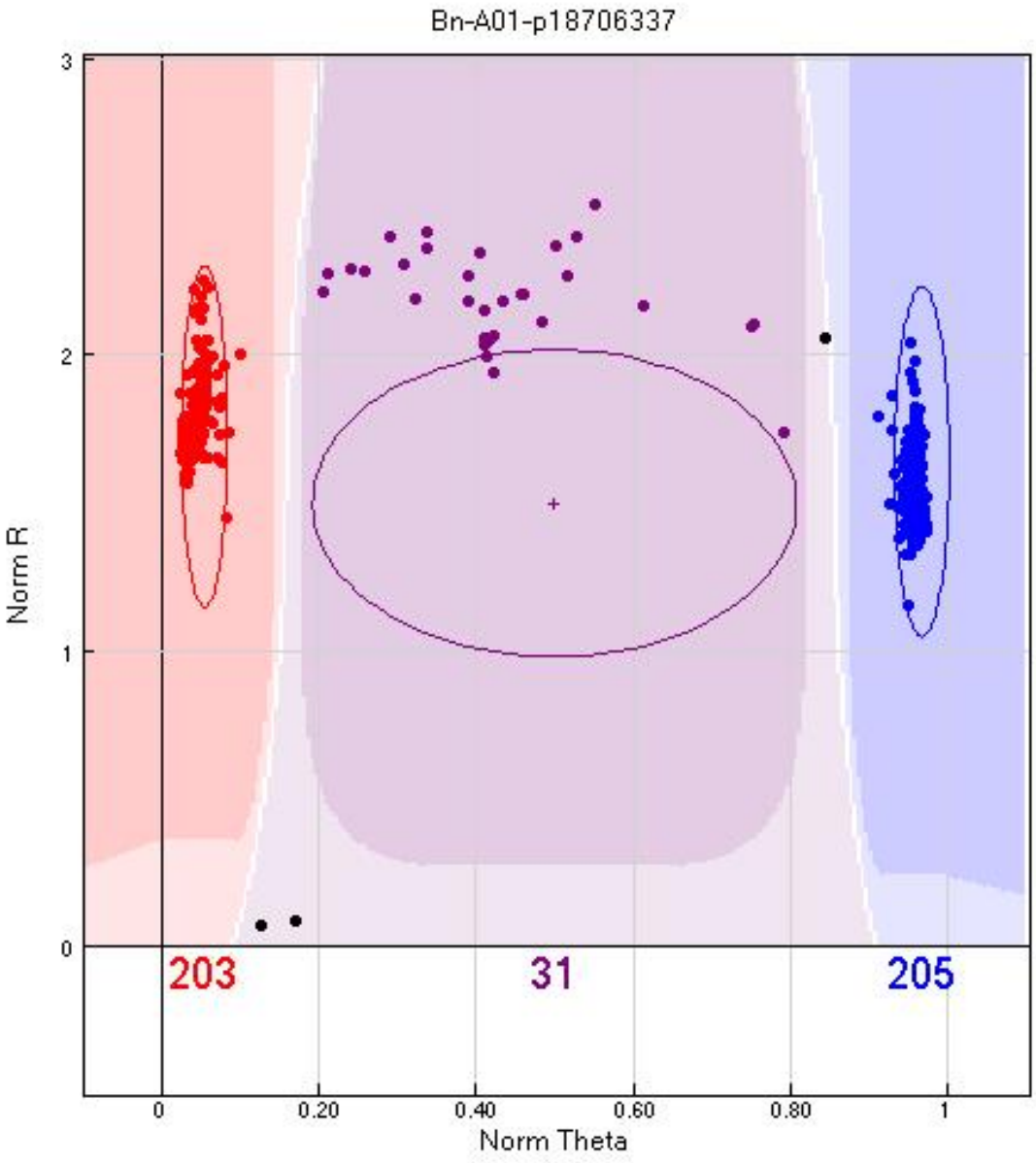

Figure 4.12: A SNP with three clear genotype clusters as called by the Illumina GenomeStudio array processing software. Each dot represents the genotype of a single individual at the SNP (Bn-A01-p18706337). Norm R is the normalized intensity of the signal and Norm Theta is the normalized theta score, where a theta score of zero indicates pure A allele signal and a theta score of one indicates pure B allele signal. Individuals in the darkly shaded region (and of the same colour) of each genotype are said to belong to that cluster.

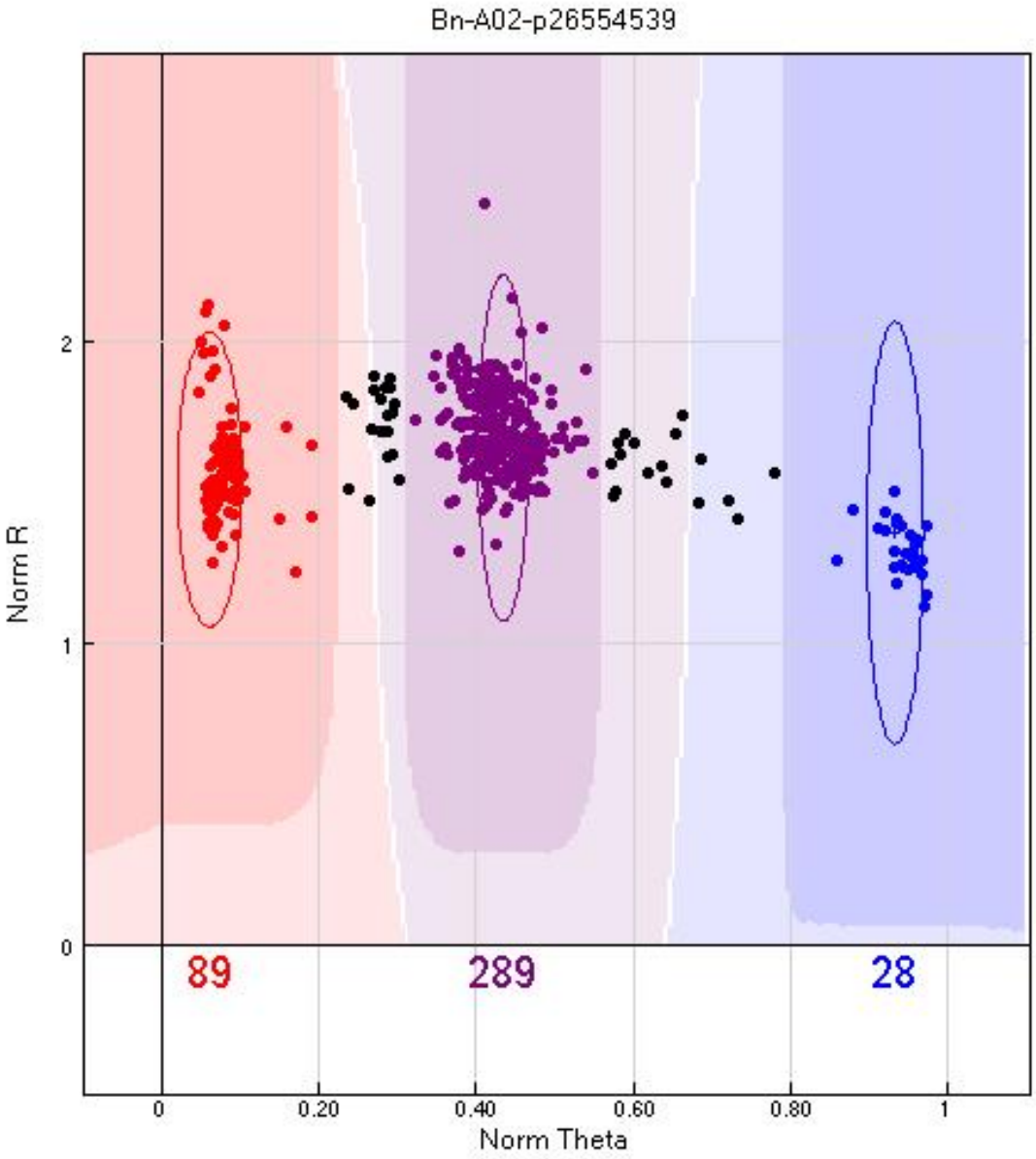

Figure 4.13: A SNP with a complex scoring pattern (5 genotype clusters) as called by the Illumina GenomeStudio array processing software. Each dot represents the genotype of a single individual at the SNP (Bn-A02-p26554539). As the GenomeStudio software was developed for use in diploid organisms, the secondary clusters of black dots are not scored. Norm R is the normalized intensity of the signal and Norm Theta is the normalized theta score, where a theta score of zero indicates pure A allele signal and a theta score of one indicates pure B allele signal. Individuals in the darkly shaded region (and of the same colour) of each genotype are said to belong to that cluster.

4.15 Conclusions

The goal of this work was to develop methods to assist in the creation of SNP genotyping arrays in non-model organisms. First a method was developed to combine per sample SNP call data (from next generation sequencing data) into a multisample format which contains sufficient biological information to provide a strong basis for the filtering of SNPs. This information includes the base call, read depth, and frequency of all alleles for each sample. It also includes determination of reference versus null positions for samples where no alternate allele has been called. As described in Section 4.4, three separate algorithms were developed, each of which combines SNP and alignment data from multiple individuals into a single output table. These algorithms, as described in Section 4.7, were evaluated for their time and space complexities. The results of the computational complexity analysis shows that all of the developed algorithms require time and space linear to the input — making them programmatically efficient. These algorithms were then implemented in Perl with differences in the implementation of each algorithm resulting in varying performance as detailed in Section 4.9. Selection of an algorithm is application specific and a discussion of suitable applications of each algorithm is given in Section 4.11.

Next, a method for filtering the multisample SNP output of the algorithms was developed, based on several biologically relevant criteria (Section 4.12). Two methods for selecting SNPs for genotyping arrays from the filtered set are discussed in Section 4.13. Section 4.13.1 describes a common method for selecting SNPs based on their distribution in the reference genome, while Section 4.13.2 describes a new approach developed in this thesis for the selection of SNPs based on matching of the Illumina probes to the reference sequences, where SNPs with probes with unique or very few matches are selected preferentially.

The efficacy of the methods described in this chapter were tested by developing several SNP genotyping arrays in both diploid and polyploid non-model organisms. Two of these genotyping arrays are described and compared to other SNP genotyping arrays in Section 4.14. The SNP filtering results presented in Section 4.14 show that the algorithms developed in this chapter and the implementations of those algorithms provide sufficient biological information to allow for the utilization of next generation sequencing SNP data for the development of SNP genotyping arrays. Comparisons to other previously published SNP genotyping arrays shows that there is a statistical improvement, when using the methods described in this chapter for the design of SNP genotyping arrays, in two key performance measures – level of polymorphism and the number of clear genotyping clusters. Further, analysis of the number of three cluster SNPs, which can be used to determine how well the preferred simple SNPs can be selected in polyploid organisms, shows that there

is a statistically significant improvement when using the SNP selection method based on probe matching described in Section 4.13.2 compared to the commonly used method of SNP selection based on distribution in the genome.

Therefore, it can be concluded, that the Brassica "6K" and "60K" SNP arrays developed using the methods described in this chapter are shown to outperform similar SNP genotyping arrays, demonstrating that the methods developed in this chapter are an improvement over current methods for the development of high-quality SNP genotyping arrays.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

During the course of this thesis several key goals have been accomplished with respect to the discovery and utilization of SNPs in non-model organisms. Chapters 1 & 2 introduced the topics of the thesis, motivation for the work, and the relevant computational and biological background. As the discovery and utilization of SNPs is highly dependent on the biology of the organism of interest, background on genome structure and organization was provided. Additionally, the modern techniques for SNP discovery described rely on the use of DNA sequencing technologies. Since the characteristics (number of reads, size of reads, common sequencing errors) of the sequencing technology used for SNP discovery can impact analysis of the results, the sequencing technologies from which data was analyzed throughout this thesis and their characteristics were described in detail. As this thesis focuses on applications to the steps prior to sequencing (Chapter 3) and after SNP discovery (Chapter 4), background on the bioinformatics of common methods for SNP discovery were also provided.

Chapter 3 describes an approach for automating the design of intron-spanning PCR primers in non-model organisms. These primers are a precursor to SNP discovery using Sanger sequencing technology and the time required for their design was a significant bottleneck in the SNP discovery process. The goal of this work was to implement an automated computational pipeline which could design PCR primers in non-model organisms by inferring the intron-exon structure of non-model genes using existing DNA sequences from the organism of interest and the gene structure from a closely related model organism as a template. The resulting PCR primers would amplify a genomic region that spanned one or more intronic regions to maximize the number of SNPs detected in subsequent analysis. Automated pipelines were implemented using Perl to combine the results of several bioinformatics utilities such as BLAST, Kalign, CAP3, Primer3, and BioPerl. Metrics for determining the success of the automated pipeline were:

1. to maintain the efficacy of PCR amplification observed in the non-automated approach,

2. decrease the manual labour required by the non-automated method,

3. decrease the time required for primer design.

Evaluation of the efficacy of the automated primer design pipeline was done using results from two primer design projects – one in which a single pair of PCR primers were design per template gene and one where multiple pairs of non-overlapping PCR primers were designed per gene template. The efficacy of primer design, as measured by the successful amplification of a PCR product in the organism of interest, ranged from 81.5% (single primer pair) to 92% (multiple primer pairs). Based on the results of a statistical test, these results show no significant difference when compared to the 88% successful amplification rate (single primer pairs only) observed for the non-automated method. Additionally, the non-automated method was estimated to require approximately 50 hours per 100 primer pairs, whereas our pipeline required only 10 minutes to design 100 PCR primer pairs. Much of the time savings are a result of the decrease in manual input required by the automated pipeline, which requires the user to provide a small number of input parameters and then fully automates the primer design process requiring no further manual intervention. The development of this pipeline resulted in the removal of a significant bottleneck in performing this type of SNP discovery and it can easily be applied to other non-model organisms.

In Chapter 4, computational methods developed to aid in the design of robust SNP genotyping arrays, specifically Illumina BeadArrays, in non-model organisms were developed. There were several goals for the work in this chapter:

1. develop and implement an algorithm for combining SNP data from multiple individuals into a single output,

2. evaluate the time and space complexity of the algorithm,

3. profile the performance of the algorithm based on different input parameters and provide an estimate of necessary hardware to process a set of inputs,

4. describe a method for filtering the resulting output of the algorithm into robust SNP markers based on biological information present in the output,

5. determine if additional methods are required to select robust SNP markers in complex genomes,

6. evaluate the approach in comparison to previously developed genotyping arrays.

Three algorithms (Serial, Parallel 1, and Parallel 2) were developed, each of which can combine SNP call information from multiple individuals with read alignment data from each individual to create a single multi-sample SNP output, and pseudocode for all three were provided. The algorithms, which differ mainly in their handling of the alignment data, were all implemented using Perl. The serial algorithm is not multi-threaded, while the parallel 1 and parallel 2 algorithms offer parallelization of the alignment processing. The implementations of the serial, parallel 1, and parallel 2 algorithms (Sections 4.5, 4.6.2, 4.6.3 respectfully) were evaluated to be linear in time (Sections 4.7.1) and space (Sections 4.7.2) complexity with respect to the input. However, their performance profiles of the Perl implementations vary significantly as discussed in Section 4.9. The parallel 1 algorithm results in the lowest running time, while the serial algorithm uses the least amount of memory and is the slowest. The memory usage of the serial and parallel 1 algorithms is dependent on the number of SNP positions and the number of individuals, while the memory usage of the parallel 2 algorithm is dependent on the reference size and the number of individuals. Selection of the appropriate algorithm is based on factors such as the size of the reference sequence set (in base pairs), the number of individuals, and the total number of unique SNP positions. Section 4.11 discusses, in depth, some strategies for selecting an appropriate algorithm given a variety of inputs as well as suggestions for methods of dealing with input data that requires more resources than the user has available.

Chapter 4 further describes a method for filtering the SNP output based on a variety of biological factors (Section 4.12.1) such as the number of lines with enough sequence depth to be considered reliable, the number of lines with heterozygous allele frequencies indicated, and the overall SNP allele frequency in the lines. Filtering SNPs with low potential to be robust markers is a primary requirement in the development of high quality SNP genotyping arrays because of the vast (potentially several million) number of SNPs that can be discovered when dealing with multiple diverse individuals sequenced using next generation sequencing technologies. Additionally, complexities in the biological makeup of the organism of interest such as repetitive elements, gene duplication and polyploidy, increase the number of markers that may result in complicated SNP genotyping analysis. In non-model organisms, Illumina does not offer matching of the probe sequences, designed from the SNP flanking sequence, back to a reference sequence set. For this reason, a solution for matching the probe sequences to the reference sequence set and selecting SNP markers based on the number of alignments of the probe sequence to the reference set was developed.

All of these methods were used in the design of a *Brassica napus* "60K" SNP genotyping array with 52,157 SNPs, while all of the methods except probe matching were used in the development of lentil, camelina, and *Brassica napus* ("6K") genotyping arrays. These applications represent both diploid (lentil) and polyploid

(camelina, *Brassica napus*) crop species and as discussed in Section 4.14, these arrays all perform (with respect to the percentage of polymorphic markers and clear cluster calls) similarly to, or outperform, arrays in species of similar genomic complexity. Specifically, the Brassica "60K" SNP array outperforms arrays in other polyploid species such as wheat and apple. Two distinct methods of SNP selection were tested: on the "60K" SNP array SNPs were selected based on matching their probe sequences to the genome and selecting SNPs based on unique or nearly unique probe matches and on the "6K" SNPs were selected based on their distribution in the genome. Observations of the performance of the Brassica "60K" array compared to that of the Brassica "6K" array indicate that the improved performance is due to selecting SNPs based on the number of matches of the Illumina probe sequences to the genome as opposed to selecting SNPs on their distribution in the genome. The improved selection of SNPs for the Brassica "60K" array seemed to provide additional robustness to the SNPs with respect to the number of clear three genotype clusters.

This thesis describes computational tools that make significant contributions to both the process of traditional SNP discovery (Chapter 3) and the utilization of discovered SNPs (Chapter 4) in non-model organisms. The work with automated PCR primer design provides a simple solution to the design of intron-spanning primers in non-model organisms where the gene structure is unknown. Additionally, while this method was intended originally for traditional SNP discovery, it has present-day applications in the area of targeted sequencing using next generation sequencing technologies. The work presented spans a technological revolution in DNA sequencing technologies resulting in increased access to large quantities of sequencing data for non-model organisms. The volume of SNP data that can be produced using next generation sequencing technologies dramatically alters the complexity of analysis, requiring innovative methods for their utilization. The implemented algorithms for combining and filtering independent multi-sample SNP data have proven successful in the design of high quality SNP genotyping platforms in several important non-model crop species. These genotyping platforms are an invaluable asset to researchers studying these species.

5.2 Future Work

Potential future applications of the automated intron-spanning primer design pipeline should focus on its applications alongside next generation sequencing technologies, as next generation sequencers offer the lowest cost per base pair sequenced. One potential application might be targeted re-sequencing of genes identified as related to traits of interest by transcriptome or reduced representation sequencing. These sequencing methods do not provide complete coverage of the genomic region. Instead, they provide data located in transcribed

regions or around restriction sites (transcriptome sequencing and reduced representation respectively). The automated primer design pipeline could be adapted to develop primers spanning these regions which would allow for the sequencing of the underlying genomic regions.

Three main areas identified for future work related to the algorithms for combining multi-sample SNP data are:

- transition to the Variant Call Format (VCF) for SNP input,
- reduction of memory usage for the parallel 1 and parallel 2 algorithms,
- development of a multi-sample SNP discovery tool.

5.2.1 Transition To The Variant Call Format (VCF) For SNP Input

During the course of this work, the VCF format has become the most widely used format for SNP output. It would therefore be beneficial to transition the developed algorithms to use this input format for SNP data from each individual. This change could be supported by modifying the subroutine called to parse SNP data to extract the required information from the VCF files. This was not included in the algorithms provided as the VCF format was not as widely used at the time of their development and the tab-delimited format available for output from the CLC Genomics Workbench provides the same information in a more human readable format.

5.2.2 Reducing Memory Usage

One potential solution for reducing the memory usage of the Perl implementation of the parallel 1 algorithm would be to clear threads upon their completion. Alignment data from each thread could be integrated into the main SNP data structure when the thread finishes and the thread could then be cleared. Clearing the thread releases the memory the thread was using which, in the case of Perl, can be significant due to copying of data structures to every thread. Currently, memory usage is proportional to the number of individuals as the memory used by each thread is held until all of the individuals have been processed. Clearing threads on termination would reduce the memory usage when the number of active threads is less than the number of individuals. Memory usage proportional to the number of active threads could result in significant savings on machines with limited resources, allowing for the use of this algorithm on a wider range of input sizes.

A further consideration with respect to reducing the memory usage is to re-implement the parallel portion of the parallel 1 algorithm in a programming language with better memory sharing amongst threads or to re-

implement the algorithm entirely in another language. As discussed in Section 4.9 the memory management for multiple threads in Perl is very poor, as it replicates data structures across all threads. Reimplementing in a language that allows for each thread to access the SNP data without having to duplicate the structure in each thread would significantly reduce the memory usage of the parallel 1 algorithm.

Memory usage of the Perl implementation of the parallel 2 algorithm might be reduced by parsing a subset of the alignment data and SNP data for each individual. The parsed alignment and SNP data from each individual could then be combined and printed to the output file. Partitioning of the alignment data into subsets could be done most easily by reference sequence, however it should be possible to generate subsets based on a specified number of reference base pairs. Both partitioning methods would allow for data to be collected for all individuals for a given reference position and combined based on the SNP positions found in all individuals. This partitioning could be accomplished by pre-generating file handles for both the SAM file and SNP file for each individual. Each thread would read the required subset of the file it was parsing and return the data. Returned SNP results would be combined across individuals and then printed to the output file using a method similar to the output algorithm described in Algorithm 4.5.6. Both partitioning methods have potential to decrease the memory usage of the parallel 2 algorithm dramatically, allowing for much larger input data sets to be processed. Unfortunately, as these methods still require storing alignment data for all positions so they cannot be applied to the parallel 1 algorithm.

The approaches described above for reducing the memory usage of the parallel 2 algorithm might allow for a user defined memory limit. In the case of partitioning input by reference sequence, the algorithm could estimate the peak memory usage based on the number of individuals in the input data and the length of the longest reference sequence. This estimation could then be compared to the the value supplied for maximum memory usage to determine if the allowed memory is sufficient. The estimation of memory usage would be based on the number of individuals in the input data and the length of the longest reference sequence. In the case of partitioning input by number of reference base pairs, the algorithm could attempt to determine the number of base pairs that could be processed based on the maximum memory and the number of individuals in the input. The estimated value would then be used to take a subset of the input data. The addition of a user defined maximum memory usage has the potential to add significantly to usability of the algorithm by providing the user a method which insures the program runs on their hardware.

5.2.3 Development Of A Multi-Sample SNP Discovery Tool

The parallel 2 algorithm has the potential to be developed into a SNP discovery tool, as the alignment data for all reads are contained in the bit vector structures. Therefore, the composition of aligned bases at a single position could be determined and compared to the nucleotide present in the reference at that position. This would remove the dependency of performing SNP discovery prior to processing the data, reducing overall analysis time. Developing this functionality would make the algorithm comparable to the GATK tool kit for multi-sample SNP calling with the added benefit of still providing the detailed information required for the design of high quality SNP genotyping arrays. If the memory improvements mentioned above for the parallel 2 algorithm were implemented, the algorithm would be able to process input for a much larger number of individuals than is currently possible.

5.2.4 Parallelization Bottlenecks

As discussed in Section 4.10, there are two main problems to explore with regards to the bottlenecks in parallelization. The first, is the efficient distribution of SAM processing jobs across the available CPU threads. This problem is referred to as the multiprocessor scheduling problem in computer science and has been shown to be NP-Complete [35]. As such, distribution of jobs should be transitioned to an implementation of a known approximate solution to the multiprocessor scheduling problem. The second, is to further investigate causes of reduced parallelization efficiency due to parallelization bottlenecks. One potential avenue of research would be to use an advanced performance profiling software such as the Intel VTune Amplifier tool, which allows for performance profiling based on direct access to the CPU [50]. Another potential avenue would be the implementation of the algorithms in another language to determine if Perl's performance is a bottleneck. Additionally, the amount of the program that is parallelized could be increased (increasing the value of P in Amdahl's law) and/or the algorithm's parallelization mechanism could be modified. One such modification would be to decouple the parallelization from the number of individuals, allowing more than one process to operate on each individual.

References

[1] 1000 Genomes Project Consortium, et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–73, Oct 2010.

[2] Teju Abegunde. Comparion of DNA Sequence Assembly Algorithms Using Mixed Data Sources. Master's thesis, University of Saskatchewan, 2010.

[3] S F Altschul, et al. Basic Local Alignment Search Tool. *J Mol Biol*, 215(3):403–10, Oct 1990.

[4] D Altshuler, et al. An SNP map of the human genome generated by reduced representation shotgun sequencing. *Nature*, 407(6803):513–6, Sep 2000.

[5] Toby Andrew, et al. Identification and Replication of Three Novel Myopia Common Susceptibility Gene Loci on Chromosome 3q26 using Linkage and Linkage Disequilibrium Mapping. *PLoS Genetics*, 4(10):e1000220, 2008.

[6] Applied Biosystems. Applied Biosystems 3730xl DNA Analyzer Specifications. `http://www6.appliedbiosystems.com/products/abi3730xlspecs.cfm`, December 2013.

[7] Nathan A Baird, et al. Rapid SNP discovery and genetic mapping using sequenced RAD markers. *PLoS One*, 3(10):e3376, 2008.

[8] W Brad Barbazuk, et al. SNP discovery via 454 transcriptome sequencing. *Plant J*, 51(5):910–8, Sep 2007.

[9] Jacqueline Batley, et al. Mining for single nucleotide polymorphisms and insertions/deletions in maize expressed sequence tag data. *Plant Physiol*, 132(1):84–91, May 2003.

[10] David R Bentley, et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–9, Nov 2008.

[11] Andreas Beyer, et al. Integrating physical and genetic maps: from genomes to interaction networks. *Nat Rev Genet*, 8(9):699–710, Sep 2007.

[12] Bioinformatics.org. IUPAC Codes. `http://www.bioinformatics.org/sms/iupac.html`, June 2014.

[13] W Burrows and D J Wheeler. A Block-sorting Lossless Data Compression Algorithm. Research Report 124, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, May 1994.

[14] Canola Council of Canada. Industry Overview. `http://www.canolacouncil.org/markets-stats/industry-overview/`, June 2014.

[15] Jun Cao, et al. Whole-genome sequencing of multiple *Arabidopsis thaliana* populations. *Nat Genet*, 43(10):956–63, Oct 2011.

[16] A C Cavell, et al. Collinearity between a 30-centimorgan segment of *Arabidopsis thaliana* chromosome 4 and duplicated regions within the *Brassica napus* genome. *Genome*, 41(1):62–9, Feb 1998.

[17] David Chagné, et al. Genome-wide SNP detection, validation, and development of an 8K SNP array for apple. *PLoS One*, 7(2):e31745, 2012.

[18] Ada Ching, et al. SNP frequency, haplotype structure and linkage disequilibrium in elite maize inbred lines. *BMC Genetics*, 3:19, October 2002.

[19] Wayne E Clarke, et al. Genomic DNA Enrichment Using Sequence Capture Microarrays: a Novel Approach to Discover Sequence Nucleotide Polymorphisms (SNP) in *Brassica napus* L. *PLoS One*, 8(12):e81992, 2013.

[20] Wayne E Clarke, et al. In preparation.

[21] Wayne E Clarke, et al. saskPrimer—An automated pipeline for design of intron-spanning PCR primers in non-model organisms. In *Bioinformatics and Biomedicine Workshops (BIBMW), 2011 IEEE International Conference on*, pages 173–177. IEEE, 2011.

[22] CLC Bio. White paper on reference assembly in CLC Assembly Cell 3.0. White paper, CLC Bio, 2010.

[23] CLC Bio. White paper on Probabilistic Variant Caller 1.1. White paper, CLC Bio, 2012.

[24] Robert Clifford, et al. Expression-based Genetic/Physical Maps of Single-Nucleotide Polymorphisms Identified by the Cancer Genome Anatomy Project. *Genome Research*, 10:1259–1265, 2000.

[25] Thomas H Cormen, et al. *Introduction To Algorithms, Volume 2*. MIT press Cambridge, 2001.

[26] Emily R A Cramer, et al. Isolation and characterization of SNP variation at 90 anonymous loci in the banded wren (*Thryothorus pleurostictus*). *Conserv. Genet.*, 9(6):1657–1660, 2008.

[27] Samuel Deutsch, et al. A cSNP Map and Database for Human Chromosome 21. *Genome Research*, 11:300–307, 2001.

[28] Andrea L Eveland, et al. Transcript profiling by 3′-untranslated region sequencing resolves expression of gene families. *Plant Physiol*, 146(1):32–44, Jan 2008.

[29] Kimberly J Felcher, et al. Integration of two diploid potato linkage maps with the potato genome sequence. *PLoS One*, 7(4):e36347, 2012.

[30] P Ferragina and G Manzini. Opportunistic Data Structures with Applications. In *41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000.

[31] National Center for Biotechnology Information. Sequence Read Archive Announcements. `http://www.ncbi.nlm.nih.gov/Traces/sra/`, January 2014.

[32] Jakob Fredslund, et al. A general pipeline for the development of anchor markers for comparative genomics in plants. *BMC Genomics*, 7:207, 2006.

[33] Jakob Fredslund, et al. PriFi: using a multiple alignment of related sequences to find primers for amplification of homologs. *Nucleic Acids Res*, 33(Web Server issue):W516–20, Jul 2005.

[34] Martin W Ganal, et al. SNP identification in crop plants. *Curr Opin Plant Biol*, 12(2):211–7, Apr 2009.

[35] Michael R Garey and David S Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[36] PK Gupta, et al. Single nucleotide polymorphisms: a new paradigm for molecular marker technology and DNA polymorphism detection with emphasis on their use in plants. *Current Science*, 80(4):524–535, 2001.

[37] John P Hamilton, et al. Single nucleotide polymorphism discovery in elite North American potato germplasm. *BMC Genomics*, 12:302, 2011.

[38] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2012.

[39] Z-L Hu, et al. Expeditor: a pipeline for designing primers using human gene structure and livestock animal EST information. *J Hered*, 96(1):80–2, 2005.

[40] Weichun Huang, et al. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–4, Feb 2012.

[41] X Huang and A Madan. CAP3: A DNA sequence assembly program. *Genome Res*, 9(9):868–77, Sep 1999.

[42] Illumina Inc. Genome Analyzer IIx Workflow and Specs. `http://www.illumina.com/systems/genome_analyzer_iix.ilmn`, June 2011.

[43] Illumina Inc. HiSeq1000 Workflow and Specs. `http://www.illumina.com/systems/hiseq_1000.ilmn`, June 2011.

[44] Illumina Inc. HiSeq2000 Workflow and Specs. `http://www.illumina.com/systems/hiseq_2000.ilmn`, May 2011.

[45] Illumina Inc. MiSeq Personal Sequencing System - Workflow and Specs. `http://www.illumina.com/systems/miseq.ilmn`, June 2011.

[46] Illumina Inc. BaseSpace: Genomics Cloud Computing. `https://basespace.illumina.com/home/sequence`, January 2012.

[47] Illumina Inc. Illumina Custom Genotyping Options. `http://www.illumina.com/documents/products/datasheets/datasheet_custom_gt.pdf`, September 2012.

[48] Illumina Inc. GenomeStudio software. `http://www.illumina.com/informatics/sequencing-microarray-data-analysis/genomestudio.ilmn`, June 2014.

[49] National Human Genome Research Institute, et al. The Human Genome Project Completion: Frequently Asked Questions. `http://www.genome.gov/11006943`, January 2012.

[50] Intel Corporation. Intel VTune Amplifier XE. `https://software.intel.com/en-us/intel-vtune-amplifier-xe`, July 2014.

[51] Sateesh Kagale, et al. The emerging biofuel crop *Camelina sativa* retains a highly undifferentiated hexaploid genome structure. *Nat Commun*, 5:3706, 2014.

[52] Robert S Kawuki, et al. Identification, characterisation and application of single nucleotide polymorphisms for diversity assessment in cassava (*Manihot esculenta* Crantz). *Molecular Breeding*, 23:669–684, 2009.

[53] W James Kent. BLAT–the BLAST-like alignment tool. *Genome Res*, 12(4):656–64, Apr 2002.

[54] Theodore Kim. Hardware-aware Analysis and Optimization of Stable Fluids. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, I3D '08, pages 99–106, New York, NY, USA, 2008. ACM.

[55] Martin Kircher and Janet Kelso. High-throughput DNA sequencing–concepts and limitations. *Bioessays*, 32(6):524–36, Jun 2010.

[56] Richard K Koehn and Thomas J Hilbish. The Adaptive Importance of Genetic Variation. *American Scientist*, 75(2):134–141, 1987.

[57] B A Konfortov, et al. Re-sequencing of DNA from a diverse panel of cattle reveals a high level of polymorphism in both intron and exon. *Mamm Genome*, 10(12):1142–5, Dec 1999.

[58] R Kota, et al. EST-derived single nucleotide polymorphism markers for assembling genetic and physical maps of the barley genome. *Funct Integr Genomics*, 8(3):223–33, Aug 2008.

[59] Robert C Lacy. Importance of Genetic Variation to the Viability of Mammalian Populations. *Journal of Mammalogy*, 78(2):pp. 320–335, 1997.

[60] E S Lander, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.

[61] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat Methods*, 9(4):357–9, Apr 2012.

[62] Ben Langmead, et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[63] Timo Lassmann and Erik L L Sonnhammer. Kalign–an accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics*, 6:298, 2005.

[64] B Lewin. *Genes VII*. Oxford University Press, 2000.

[65] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–60, Jul 2009.

[66] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–95, Mar 2010.

[67] Heng Li, et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–9, Aug 2009.

[68] Heng Li, et al. SAM tools Beta Release 0.1.9. `http://sourceforge.net/mailarchive/forum.php?forum_name=samtools-announce&max_rows=75&style=nested&viewmonth=201010&viewday=28`, October 2010.

[69] Heng Li, et al. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–8, Nov 2008.

[70] Ruiqiang Li, et al. The sequence and de novo assembly of the giant panda genome. *Nature*, 463(7279):311–7, Jan 2010.

[71] Ruiqiang Li, et al. SNP detection for massively parallel whole-genome resequencing. *Genome Res*, 19(6):1124–32, Jun 2009.

[72] Ruiqiang Li, et al. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–4, Mar 2008.

[73] Ruiqiang Li, et al. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–7, Aug 2009.

[74] Diego Lijavetzky, et al. High throughput SNP discovery and genotyping in grapevine (*Vitis vinifera* L.) by combining a re-sequencing approach and SNPlex technology. *BMC Genomics*, 8:424, 2007.

[75] De Ma, et al. Performance Estimation Techniques With MPSoC Transaction-Accurate Models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(12):1920–1933, 2013.

[76] M Mahner and M Kary. What exactly are genomes, genotypes and phenotypes? And what about phenomes? *J Theor Biol*, 186(1):55–63, May 1997.

[77] Marcel Margulies, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–80, Sep 2005.

[78] G T Marth, et al. A general approach to single-nucleotide polymorphism discovery. *Nat Genet*, 23(4):452–6, Dec 1999.

[79] Peter J Maughan, et al. SNP Discovery via Genomic Reduction, Barcoding, and 454-Pyrosequencing in Amaranth. *The Plant Genome*, 2(3):260–269, Nov 2009.

[80] A McKenna, et al. GATK DepthPerAlleleBySample Release Note. `https://gatkforums.broadinstitute.org/discussion/1028/depthperallelebysample`, July 2012.

[81] A McKenna, et al. GATK VariantFiltration. `http://www.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_sting_gatk_walkers_filters_VariantFiltration.html`, June 2014.

[82] Aaron McKenna, et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*, 20(9):1297–303, Sep 2010.

[83] N J D Nagelkerke. A Note on a General Definition of the Coefficient of Determination. *Biometrika*, 78(3):pp. 691–692, 1991.

[84] S B Needleman and C D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–53, Mar 1970.

[85] Jason O'Rawe, et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. *Genome Med*, 5(3):28, Mar 2013.

[86] Isobel A P Parkin, et al. Towards unambiguous transcript mapping in the allotetraploid *Brassica napus*. *Genome*, 53(11):929–38, Nov 2010.

[87] Isobel A P Parkin, et al. Transcriptome and methylome profiling reveals relics of genome dominance in the mesopolyploid *Brassica oleracea*. *Genome Biol*, 15(6):R77, 2014.

[88] W R Pearson and D J Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85(8):2444–8, Apr 1988.

[89] Geo Pertea, et al. TIGR Gene Indices clustering tools (TGICL): a software system for fast clustering of large EST datasets. *Bioinformatics*, 19(5):651–2, Mar 2003.

[90] Antoni J Rafalski. Applications of single nulceotide polymorphisms in crop genetics. *Current Opinion in Plant Biology*, 5:94–100, 2002.

[91] C Ramel. Mini- and microsatellites. *Environ Health Perspect*, 105 Suppl 4:781–9, Jun 1997.

[92] Vasily Ramensky, et al. Human non-synonymous SNPs: server and survey. *Nucleic Acids Res*, 30(17):3894–900, Sep 2002.

[93] Roche Diagnostics Corporation. 454 Sequencing - GS Junior- Instrument and Workflow. `http://www.gsjunior.com/instrument-workflow.php`, June 2011.

[94] Roche Diagnostics Corporation. 454 Sequencing System, How it Works. `http://454.com/products-solutions/how-it-works/index.asp`, May 2011.

[95] Roche Diagnostics Corporation. 454 Sequencing System, System Features. http://454.com/products-solutions/system-features.asp, May 2011.

[96] S Rozen and H Skaletsky. Primer3 on the WWW for general users and for biologist programmers. *Methods Mol Biol*, 132:365–86, 2000.

[97] P J Russell. *iGenetics: A Molecular Approach*. The Genetics Place Series. Pearson/Benjamin Cummings, 2006.

[98] F Sanger, et al. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A*, 74(12):5463–7, Dec 1977.

[99] Andrew G Sharpe, et al. Ancient orphan crop joins modern era: gene-based SNP discovery and mapping in lentil. *BMC Genomics*, 14:192, 2013.

[100] Christine Sidebottom. Personal Communication, 2008.

[101] Ravinder Singh, et al. In preparation.

[102] Andrew D Smith, et al. Updates to the RMAP short-read mapping software. *Bioinformatics*, 25(21):2841–2, Nov 2009.

[103] Andrew D Smith, et al. Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics*, 9:128, 2008.

[104] Lloyd M Smith, et al. Fluorescence detection in automated DNA sequence analysis. *Nature*, 321:674–679, 1986.

[105] T F Smith and M S Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–7, Mar 1981.

[106] LGC Genomic Solutions. How does KASP work. http://www.lgcgenomics.com/genotyping/kasp-genotyping-chemistry/how-does-kasp-work/, September 2014.

[107] LGC Genomic Solutions. KASP Overview. http://www.lgcgenomics.com/genotyping/kasp-genotyping-chemistry/kasp-overview/, September 2014.

[108] P Stam. Construction of integrated genetic linkage maps by means of a new computer package: Join Map. *The Plant Journal*, 3(5):739–744, 1993.

[109] Frank J Steemers and Kevin L Gunderson. Whole genome genotyping technologies on the BeadArray platform. *Biotechnol J*, 2(1):41–9, Jan 2007.

[110] Michael Strömberg and Wan-Ping Lee. Mosiak Assembler. http://bioinformatics.bc.edu/marthlab/Mosaik, September 2012.

[111] Dan Sugalski and Nicholas Clark. Devel::Size - Perl extension for finding the memory usage of Perl variables. http://search.cpan.org/~nwclark/Devel-Size-0.79/lib/Devel/Size.pm, June 2014.

[112] H Swerdlow and R Gesteland. Capillary gel electrophoresis for rapid, high resolution DNA sequencing. *Nucleic Acids Res*, 18(6):1415–9, Mar 1990.

[113] The SAM/BAM Format Specification Working Group. Sequence Alignment/Map Format Specification. http://samtools.github.io/hts-specs/SAMv1.pdf, December 2013.

[114] Martin Trick, et al. Single nucleotide polymorphism (SNP) discovery in the polyploid *Brassica napus* using Solexa transcriptome sequencing. *Plant Biotechnol J*, 7(4):334–46, May 2009.

[115] Michela Troggio, et al. Evaluation of SNP Data from the Malus Infinium Array Identifies Challenges for Genetic Analysis of Complex Genomes of Polyploid Origin. *PLoS One*, 8(6):e67407, 2013.

[116] Harm van Bakel, et al. The draft genome and transcriptome of *Cannabis sativa*. *Genome Biol*, 12(10):R102, 2011.

[117] Jan van Oeveren, et al. Sequence-based physical mapping of complex genomes by whole genome profiling. *Genome Res*, 21(4):618–25, Apr 2011.

[118] Rajeev K Varshney, et al. Genic microsatellite markers in plants: features and applications. *Trends Biotechnol*, 23(1):48–55, Jan 2005.

[119] Ignazio Verde, et al. Development and evaluation of a 9K SNP array for peach by internationally coordinated SNP detection and validation in breeding germplasm. *PLoS One*, 7(4):e35668, 2012.

[120] Shichen Wang, et al. Characterization of polyploid wheat genomic diversity using a high-density 90 000 single nucleotide polymorphism array. *Plant Biotechnol J*, Mar 2014.

[121] Xiaowu Wang, et al. The genome of the mesopolyploid crop species *Brassica rapa*. *Nat Genet*, 43(10):1035–9, Oct 2011.

[122] Andreas P M Weber, et al. Sampling the Arabidopsis transcriptome with massively parallel pyrosequencing. *Plant Physiol*, 144(1):32–42, May 2007.

[123] Stefan Weckx, et al. SNPbox: web-based high-throughput primer design from gene to genome. *Nucleic Acids Res*, 32(Web Server issue):W170–2, Jul 2004.

[124] Kris Wetterstrand. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). `http://www.genome.gov/sequencingcosts`, December 2013.

[125] M Woodhouse, et al. Polyploidy. *Nature Education*, 2(1), 2009.

[126] Jianbing Yan, et al. Genetic Characterization and Linkage Disequilibrium Estimation of a Global Maize Collection Using SNP Markers. *PLoS ONE*, 4(12):e8451, 12 2009.

[127] Frank M You, et al. ConservedPrimers 2.0: a high-throughput pipeline for comparative genome referenced intron-flanking PCR primer design and its application in wheat SNP discovery. *BMC Bioinformatics*, 10:331, 2009.

[128] Marcus A Zachariah, et al. A generalized affine gap model significantly improves protein sequence alignment accuracy. *Proteins*, 58(2):329–38, Feb 2005.

Appendix A

IUPAC Ambiguity Codes

IUPAC Ambiguity Code	Base
R	A or G
Y	C or T
S	G or C
W	A or T
K	G or T
M	A or C
B	C or G or T
D	A or G or T
H	A or C or T
V	A or C or G
N	A or C or G or T

Table A.1: IUPAC codes codes for ambiguous bases