Chapter 1

Introduction

A hallmark of modern science is the collection of large volumes of data by measuring physical processes. Experiments which collect terabytes of data are common in particle physics and astronomy. In biology, we can now sequence the DNA of previously uncharacterized organisms, entire human populations or thousands of human cancers. These applications need extremely computationally efficient algorithms to process the data. In this work, I will present efficient algorithms for processing raw DNA sequence data. The algorithms I will develop are based on the idea of querying a *compressed data structure*. These data structures become more efficient in memory use as the redundancy in the data increases, while retaining the ability to perform efficient queries. Throughout this text I will develop algorithms for building and querying these structures in the context of DNA sequence data. Using the algorithms I develop, I will address two major problems in sequence analysis. The first problem is the efficient reconstruction of a genome - the full complement of DNA within a cell - using only the output from a DNA sequencing instrument. Given the scale of the problem, often requiring hundreds of gigabytes of data, computation time and memory efficiency is a primary concern. The second problem that I will address is the detection of differences in the sequence of DNA between related genomes. The approach I will develop does not require the alignment of raw sequence reads to a reference genome - it works directly from the sequence reads themselves.

In the remainder of this chapter, I will introduce DNA sequencing and the key problem of reconstructing a genome from a set of sequence reads. I will

also discuss previous approaches to the reconstruction problem and give a brief overview of compressed data structures. At the end of this chapter, I provide an overview of the remainder of this text.

1.1 DNA Sequencing

The field of molecular biology developed rapidly in the second half of the 20th century. Watson and Crick's discovery of the double-helix structure of DNA told us how genetic information is copied within a cell and transmitted from generation to generation. The determination of the genetic code - how the sequence of nucleotides in genes encode the information necessary for constructing proteins - led to the formation of the central dogma of molecular biology, which describes how DNA encodes RNA which encodes proteins. It was thus readily apparent that the sequence of nucleotides making up an individual's genome underlies the protein complement of the cell and hence much phenotypic variation that we see between individuals. The development of methods for determining the sequence of nucleotides in DNA molecules would have great importance to the study of human health and the evolution of life. Using techniques developed by Sanger and colleagues to determine the nucleotide sequence of short RNA fragments [Sanger et al., 1965], Walter Fiers and colleagues sequenced the first gene, the coat protein of bacteriophage MS2 [Jou et al., 1972]. Later in 1976 the complete sequence of bacteriophage MS2 would be determined [Fiers et al., 1976]. Maxam and Gilbert developed a method for directly determining the sequence of nucleotides in DNA based on breaking radioactively-labelled DNA at specific positions, then size-sorting the DNA fragments using gel electrophoresis [Maxam and Gilbert, 1977]. Also in 1977 Sanger, Nicklen and Coulson developed a method that would dominate DNA sequencing for almost three decades. This method, called chain-termination sequencing and widely known as Sanger sequencing, is based on introducing specially modified nucleotides which do not allow extension of a DNA chain. When these terminating nucleotides are incorporated into a growing DNA chain, they stop the reaction from proceeding any further. The result is a mixture of partial copies of the original template DNA. This mixture can be sorted by fragment length using gel electrophoresis, and the sequence of

the DNA template can be read from the gel by the pattern of bands indicating which modified base (A, C, G or T) stopped the chain at a given position [Sanger et al., 1977]. Sanger and Gilbert's contributions to DNA sequencing would earn them the 1980 Nobel Prize in Chemistry (shared with Paul Berg).

Over the next three decades the efficiency and throughput of chain-termination sequencing was greatly improved, particularly as a result of automation of the sequencing process. Gel electrophoresis and radioactively labelled nucleotides were replaced by capillary tubes and fluorescent nucleotides, allowing the automated imaging and analysis of the sequencing reaction using a computer [Smith et al., 1986]. Multiple sequencing reactions were run in parallel. The read length - the number of contiguous bases sequenced in a single reaction - improved to 500-1000 bases. These improvements to the throughput of DNA sequencing led to the start of the genomics era, where entire genomes could be sequenced. The first complete genome sequenced of a free-living organism was the 1.8 megabase genome of *Haemophilus influenzae* published in 1995 [Fleischmann et al., 1995]. As the cost of sequencing continued to fall, larger genomes were sequenced like that of *Escherichia coli* [Blattner et al., 1997], *Saccharomyces cerevisiae* [Goffeau et al., 1996], *Caenorhabditis elegans* [C. elegans Sequencing Consortium, 1998] and the model plant *Arabidopsis thaliana* [Arabidopsis Genome Initiative, 2000].

An international consortium to sequence the human genome was formed in the early 1990s. A competing privately funded project was later started by the Celera corporation. These competing projects progressed in contrasting styles. As the human genome was already known to be highly repetitive [Schmid and Deininger, 1975], the publicly funded Human Genome Project (HGP) took a conservative approach to sequencing. They developed libraries of Bacterial Artificial Chromosomes (BACs), 150 kilobase fragments of human DNA copied in a bacterial cell. Restriction digestions of the BACs were created and ordered into a 'map' based on the overlapping patterns of restriction fragments formed by gel electrophoresis. Using the map, individual BACs were selected for direct sequencing to tile across each chromosome. The BAC map gave a scaffold on which to place the individually sequenced clones. The privately funded project opted for a more aggressive, faster approach. This method, termed *whole genome shotgun sequencing*, did not construct a map but rather sampled random sequence

reads from the entire genome. These raw shotgun reads would be augmented by 'mate-pair' reads where both ends of a long DNA fragment would be sequenced, with unknown sequence in between. This strategy relied on the development of sophisticated computational algorithms to determine the order of the sequence reads and assemble them into the genome. In 2001, the two projects published their drafts of the human genome [International Human Genome Sequencing Consortium, 2001; Venter et al., 2001]. A vigorous debate on the effectiveness and independence of Celera's approach to sequencing the human genome ensued in the literature for a number of years [Adams et al., 2003; Green, 2002; Myers et al., 2002; Waterston et al., 2002, 2003]. Subsequent to the sequencing of the human genome, the genome of a laboratory strain of mouse was sequenced using a combination of BAC-based and whole genome shotgun data [Mouse Genome Sequencing Consortium, 2002].

More recently many genomes have been sequenced including the Chimpanzee [Chimpanzee Sequencing and Analysis Consortium, 2005], the Giant Panda [Li et al., 2010a], the Gorilla [Scally et al., 2012] and the Bonobo [Prufer et al., 2012]. The default is now whole genome shotgun assembly, which can provide the complete genome sequence for small prokaryotic genomes but for larger and more complex genomes assemblies are typically incomplete.

1.1.1 High Throughput Sequencing

While the efficiency of Sanger sequencing was improved by orders of magnitude since its conception, the cost of sequencing remained too high for the routine sequencing of entire human genomes. A second generation of sequencing technology arose in the mid-2000s, based on performing and measuring millions of sequencing reactions in parallel. These technologies are collectively referred to as High Throughput Sequencing (HTS) or Next Generation Sequencing (NGS).

The first HTS technology developed is termed "pyrosequencing" and was commercialized by 454 Life Sciences[1]. In this method of sequencing, single-stranded DNA is captured by beads, amplified and loaded into picoliter reaction wells. Fluorescently labelled nucleotides are added to the reaction in a predefined or-

[1]Later acquired by Roche

der. When a labelled base is bound to the template DNA, a short pulse of light is released which can be detected with a CCD camera. After each reaction, the reagents are cleared before the next addition of bases. The captured images are analyzed in real time to determine the sequence of each template molecule [Wheeler et al., 2008]. This method of sequencing produces far more data per run than Sanger sequencing, generating up to 700 megabases of sequence per 23 hour run, with up to 1000bp read lengths[1].

A second massively parallel sequencing technology was developed from work begun by Balasubramanian and Klenerman at the University of Cambridge and subsequently commercialized by Solexa[2]. In this method, template DNA is ligated to sequences fixed on a slide. The template DNA is amplified in place to generate a cluster of molecules. The sequencing process occurs over a number of cycles. In each cycle reversibly-terminated nucleotides, each labelled with a fluorescent dye, is added to the reaction. An image is taken, then the dye and terminator are chemically removed to allow the reaction to proceed to the next base in the chain. The captured images are analyzed after the run, and the identity of which base was incorporated during each cycle is determined by the color of each cluster [Bentley et al., 2008]. This method now produces up to 600 gigabases per run, when 100bp reads are taken from both ends of a DNA fragment[3].

Other approaches include sequencing-by-ligation (SOLiD by Life Technologies, first used in Valouev et al. 2008, and Complete Genomics Drmanac et al. 2010). Recently single molecule methods requiring no DNA amplification have become available (PacBio, Eid et al. 2009). In principle these can give very long reads, beyond the effective limit of 1000bp for preceding technologies, but currently they are not cost and accuracy competitive.

The enormous volume of data generated by HTS instruments has allowed population surveys of human genome variation [1000 Genomes Project Consortium, 2010] and plans to sequence thousands of human cancers [The International Cancer Genome Consortium, 2010] and 10,000 vertebrate genomes *de novo* [Genome 10K Community of Scientists, 2009]. High throughput sequencing has also gen-

[1]This information was taken from http://454.com/ on July 16th, 2012
[2]Later acquired by Illumina, Inc
[3]This information was taken from http://www.illumina.com on July 16th, 2012

erated new analysis challenges. As whole genome sequencing is now a routine experimental measure, we need algorithms and software that can scale to match the data generated. This is particularly important for the computationally demanding *de novo* assembly problem.

1.2 Sequence Assembly

Even at the earliest stages of DNA sequencing, when the genomes sequenced were only a few kilobases in length, it was apparent that computers would be needed to help analyze the data. In [Staden, 1979] Roger Staden observes that "It became clear during the sequencing of bacteriophage ϕX174 DNA that it was necessary to use computers to handle and analyze the data". As sequencing technology has progressed over the last 30 years, computational techniques to analyze the data have developed in parallel. One of the fundamental computational problems in DNA sequence analysis is the reconstruction of a genome from a set of sequence reads. This problem is known as *de novo* assembly. For large genomes, billions of reads may be used in the assembly and the time and space efficiency of the algorithm is crucial. In this section we give an overview of assembly algorithms. These will be discussed in more technical detail in the following chapter.

Early assemblers were not fully automated but helped a user identify and merge overlapping sequence reads. The Staden package referenced above is an example of this approach. As the size of sequence data sets grew, fully automated assemblers needed to be developed. Early assemblers often used a greedy algorithm. Pairs of reads would be compared to find overlaps and each overlap would be scored based on the number of matching and mismatching bases. The reads sharing the highest scoring overlap would be merged together and the process would iterate. As this process made a greedy choice, care needed to be taken to avoid merging reads that originate from similar repeats. The Phrap program[1], designed for assemblies of BACs and used by the Human Genome Project, used base quality scores[2] to help distinguish between true overlaps and those caused by identical or nearly identical repeats.

[1] Unpublished, algorithm described here http://www.phrap.org/phredphrap/phrap.html
[2] An estimation of the probability that a given nucleotide in the read is incorrect

In a move away from greedy algorithms, Kececioglu and Myers modelled the assembly problem as a variant of the Shortest Common Superstring problem, which was known to be NP-Hard [Kececioglu and Myers, 1993]. Their formulation of the problem used the concept of an overlap graph. In an overlap graph, each sequence read is a vertex and two vertices are connected by an edge if their corresponding reads have a significant overlap. The assembly problem is thus to find walks through the graph that are consistent with the overlap relationships. A final consensus sequence can be computed from a multiple alignment constructed from the set of overlapping reads implied by the walk. The three stages of assembly - overlap computation, layout, consensus gave rise to the "OLC" acronym used to describe assemblers following this paradigm. The overlap computation stage is typically the computational bottleneck in the assembly. In the worst case this requires $O(N^2)$ time where N is the total number of bases in the sequence reads. In the Celera assembly of the human genome, they compared all reads against each other, which required 10,000 CPU hours on a cluster of 40 machines [Venter et al., 2001]. In [Myers, 2005], Gene Myers reformulated overlap graph assembly in terms of *string graphs*. Myers' string graph construction algorithm removes transitive edges from an overlap graph. As this requires the full overlap graph to be constructed it shares the same computational bottleneck as OLC assembly. Various strategies to accelerate overlap detection have been developed, including limiting overlap computation to pairs of reads that have an exact, short match [Rasmussen et al., 2006]. Despite such improvements, overlap computation remained a significant bottleneck. This became particularly important when High Throughput Sequencing instruments became widely available, as traditional overlap-based strategies could not cope with the volume of data.

In the late 1980s, a new approach to DNA sequencing was proposed in which DNA is hybridized to an array containing short oligonucleotide probes with known sequences. A signal is read from each probe, indicating whether or not the particular sequence is present in the genome. Thus, the assembly problem is to reconstruct the genome from the *spectrum* of short sequences that it contains. This approach to DNA sequencing, called Sequencing by Hybridization, did not have a significant impact on the *de novo* sequencing of genomes but in studying this problem a new class of assembly algorithms was developed, pioneered by

Pavel Pevzner [Pevzner, 1989; Pevzner et al., 2001] along with Idury and Waterman [Idury and Waterman, 1995]. The defining characteristic of these algorithms is that they break sequence reads into chains of consecutive k-mers[1], overlapping by $(k-1)$ bases, and construct a graph of the relationship between k-mers. Such assembly graphs are called *de Bruijn graphs* after the graphs used by Nicolaas de Bruijn to study combinatorial problems [de Bruijn, 1946]. As the construction of a de Bruijn graph only requires performing exact matches between k-mers, the construction can be performed in linear time using a hash table. The straightforward construction of the graph, along with the efficiency in which repetitive sequences are handled by the graph, led to the de Bruijn graph becoming the dominant method of assembly for high throughput, short read sequence data. This approach to assembly was initially applied to HTS data by Chaisson and Pevzner [Chaisson and Pevzner, 2008] and Zerbino and Birney [Zerbino and Birney, 2008].

While the de Bruijn graph approach to assembly solved the computation time problem, the amount of memory required to store the graph became a major concern. In the de Bruijn graph, there is a vertex for every unique k-mer in the genome. In addition, sequencing errors cause new, erroneous k-mers to be added. The de Bruijn graph of large genomes can have billions of vertices, requiring hundreds of gigabytes of memory. Reducing the memory requirements of de Bruijn graph assembly is a very active area of research. Simpson et al. [2009] designed a representation of the de Bruijn graph which could be distributed across a network of computers, spreading the memory load across multiple machines. Li et al. [2010c] performed error correction before assembly to reduce the number of vertices in the graph. Conway and Bromage proposed encoding the structure of the graph using sparse bit vectors [Conway and Bromage, 2011]. Recently Pell et al. have developed a probabilistic representation of a de Bruijn graph using a bloom filter [Pell et al., 2012]. Chikhi and Rizk have also recently used a bloom filter to represent a de Bruijn graph [Chikhi and Rizk, 2012].

[1] subsequences of a uniform length, k

1.2.1 A Practical Overview of Assembly

The input into a genome assembly is a set of sequence reads from the genome of interest. Often *paired-end* reads will be obtained. In paired-end sequencing, both ends of the DNA fragment will be read without reading the sequence between the ends. For example the first and last 100 bases of a 500 base fragment of DNA may be read - the 300bp sequence separating the ends is unknown. *Mate-pair* reads may also be obtained. In mate-pair sequencing a multi-kilobase fragment of DNA is circularized then cut twice, and the ends from the two cut points are read. This allows a wider separation between the sequenced pair, up to 3 to 10 kilobases. When discussing sequencing data, I will refer to the *coverage* of the genome. This is a measure of how redundantly the genome is sampled by the reads. For example sequence coverage of 40X indicates that on average each base of the genome is represented in 40 reads. This is often also called the sequencing *depth*.

The assembler will read in all available data and output a set of *contigs* and *scaffolds*. The contigs are the primary output of the assembly. These are stretches of the genome that have been completely assembled from the raw reads. They contain no gaps. When the assembler finds a repeat that it cannot resolve, or a contig cannot be extended due to a lack of reads covering the genome, the contig assembly stops. If paired-end or mate-pair reads are available the assembler can build scaffolds from the contigs. The paired reads provide long-range information that can jump over the coverage gap or unresolvable repeat. The scaffolds will contain multiple contigs separated by gaps. The gaps will be encoded using ambiguity symbols (typically runs of "N" symbols) which estimates the length of the unresolved sequence. The lengths of the scaffolds will typically be much greater than the lengths of the input contigs.

To measure the quality of assembly the *N50* length of the contigs or scaffolds can be calculated. The N50 of a set of contigs or scaffolds is the length x such that contigs of length x or greater contain half of the total length of the assembly. Misassembled contigs may inflate the N50 length of an assembly. If a reference genome is available we may calculate NG50 instead - this calculates the N50 length of segments of the genome that have been correctly assembled, which accounts

for the possibility of misassembled contigs or scaffolds [Earl et al., 2011].

1.2.2 The Topology of Assembly Graphs

The structure of the assembly graph can give important information about the underlying genome. Here I will describe common topological features of assembly graphs. The features below are common to both de Bruijn graphs and string graphs. Any distinctions will be noted.

1.2.2.1 Graph Tips

On the Illumina sequencing platform, errors are more likely to occur at the end of a read. When constructing a de Bruijn graph from the k-mers of a read containing sequencing errors, the k-mers covering the position of the error are typically unique. These k-mers will form a chain of vertices in the de Bruijn graph that terminates with the last k-mer in the read. As these branches of the graph are only connected on one end, they are typically termed *tips* of the graph. A simple tip is depicted in figure 1.1. In the string graph, these structures can also form. In this case, reads with sequencing errors will not have a suffix (or prefix) overlap. This will break the chain of overlapping reads, leading to a disconnected branch in the graph. The tips in a string graph will typically be shorter than those of a de Bruijn graph, as in the de Bruijn graph a single sequencing error may generate multiple erroneous vertices.

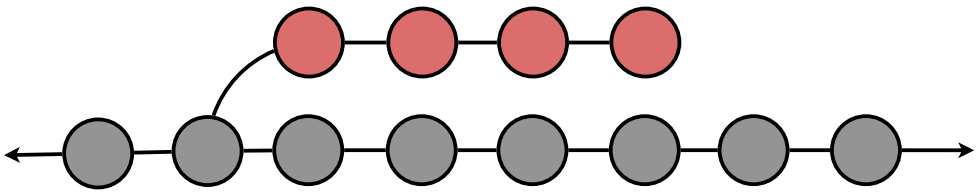

Figure 1.1: A simple tip in an assembly graph. The red vertices contain sequencing errors - due to these errors the sequence of this branch diverges from the rest of the graph (grey vertices). The arrows on the terminal grey vertices are to indicate the graph continues off-page.

A standard graph cleaning operation found in most graph-based assemblers is to identify these tips then trim them back to the point of divergence. In the example given in 1.1 this would remove the four red vertices, which would remove the ambiguity in the edge set of the second grey vertex.

1.2.2.2 Graph Bubbles

When the genome contains nearly-identical sequences structures known as *bubbles* form in the graph. When sequencing a diploid genome bubbles will form around heterozygous variants. The k-mers (or reads) covering the variants will cause a branch in the graph, with two possible paths to follow. The distinguishing feature of the bubble is that these two paths will collapse back together a short distance later. An example of a bubble is shown in figure 1.2.

Bubbles can also form around recurrent sequencing errors or inexact copies of repeats dispersed throughout a genome. As with tip removal, removing bubbles is a common operation performed on assembly graphs. Typically one of the two halves of the bubble will be deleted from the graph. A bubble removal algorithm is described in section 3.2.5. Detecting these structures in the graph will form the basis of the algorithms presented in Chapter 4.

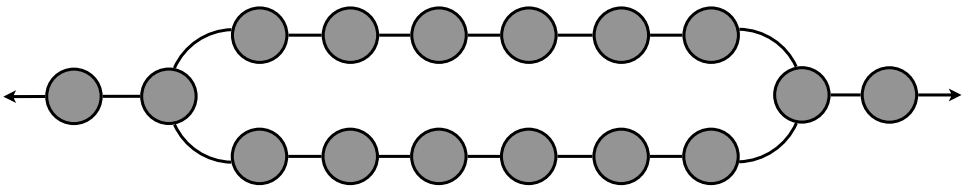

Figure 1.2: A bubble in the graph showing the distinctive divergence/collapsing signature.

1.2.2.3 Repeats

Genomes contain identical or nearly-identical *repeat* sequences. These sequences cause ambiguity in the assembly graph. Figure 1.3 depicts the simplest possible situtation where there are two exact copies of a repeat in a genome. The red nodes in this example graph indicate the repetitive sequence. The repeat segment of

the graph has two entry and exit points. This indicates there are four possible paths through this segment of the graph - *XRW*, *XRZ*, *YRW*, *YRZ* - only two of which are correct. In the absence of all other information we cannot resolve this repeat. When working with a de Bruijn graph, we may try to thread the original sequence reads through the graph in an attempt to resolve the repeat. If we can find reads that contain both X and W or X and Z it will help us determine the correct pair of paths traversing this segment of the graph. If the repeat is very long or has many copies in the genome, it is unlikely that it can be easily resolved. Paired end or mate pair data can be used to help resolve these cases.

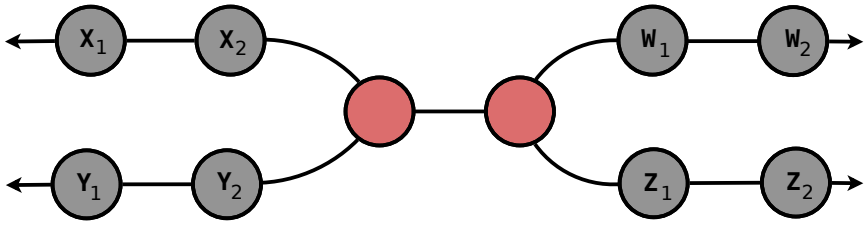

Figure 1.3: A simple repeat in the assembly graph. The red nodes represent sequences present multiple times in the genome.

1.2.2.4 Unipaths

Vertices in the graph that are connected to only one neighbor are *unambiguous*. Such vertices can be merged together into a single vertex without the loss of information or the possibility of making a misassembly. Chains of unambiguous vertices are referred to as *unipaths*. Finding unipaths is the primary method of *contig* assembly in most graph based assemblers. Figure 1.4 shows the unipath graph of the simple repeat from figure 1.3.

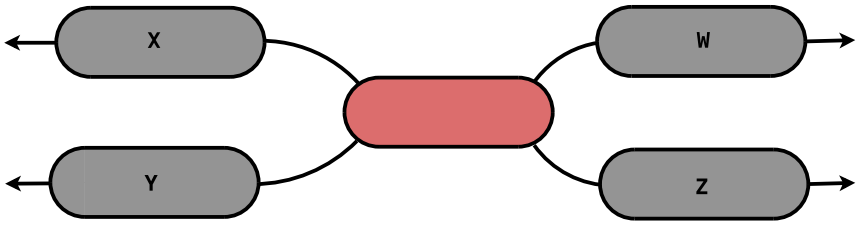

Figure 1.4: A unipath graph constructed from the graph depicted in figure 1.3. The unambiguously connected vertices have been merged together.

1.2.2.5 Assembly Software

The development of short read assemblers has been one of the most active areas of bioinformatics since the introduction of high-throughput sequencing. SSAKE was one of the first assemblers designed specifically for short reads [Warren et al., 2007]. The Velvet assembler [Zerbino and Birney, 2008] was introduced shortly afterwards and became very popular for assembling small-to-medium sized genomes. Velvet uses an explicit representation of the de Bruijn graph based on memory pointers in the C programming language. This representation of the graph requires large amounts of memory when the de Bruijn graph has many vertices, like in the case of assembling a human genome. The ABySS assembler was designed specifically for large genomes [Simpson et al., 2009]. ABySS does not use an explicit pointer-based de Bruijn graph, rather it stores a collection of k-mers in a sparse hash table from which the structure of the graph can be inferred. This representation of the graph allows the memory load to be distributed over a cluster of computers. ABySS was used in the Gorilla [Scally et al., 2012] and Tomato genome projects [Tomato Genome Consortium, 2012]. The SOAPdenovo assembler [Li et al., 2010c] was also designed for large genomes and reduces memory consumption by first performing k-mer based error correction to avoid adding erroneous k-mers to the de Bruijn graph. SOAPdenovo was one of the first short read assemblers to focus on the use of multiple large-insert mate-pair libraries for scaffold construction. This technique was used in the Giant Panda genome project [Li et al., 2010a]. The ALLPATHS series of assemblers is designed by the Broad Institute [Butler et al., 2008; Gnerre et al., 2011; Maccallum

et al., 2009]. ALLPATHS is unique in that it requires specific types of sequencing reads[1]. The algorithms are optimized for this input, allowing high-quality assemblies to be generated when the requirements are met [Gnerre et al., 2011]. In the Assemblathon competition [Earl et al., 2011], ALLPATHS-LG and SOAPdenovo were the top two entries out of seventeen groups who submitted assemblies. The assembler described in Chapter 3 of this thesis, SGA, was ranked third. ABySS was seventh.

The computational requirements for these popular assemblers varies widely. Velvet is very fast but cannot run on large data sets. For a human genome, ABySS requires 150-400GB of aggregate memory across the assembly cluster. SOAPdenovo requires a similar amount of memory after the error correction step. The authors of ALLPATHS-LG suggest the use of a multi-core 512GB server and estimate the run time to be 3.5 weeks for a human genome [Gnerre et al., 2011].

New assemblers have been developed recently with a focus on memory efficiency. Cortex [Iqbal et al., 2012] uses an approach similar to ABySS and encodes a de Bruijn graph using an efficient hash table of k-mers. Gossamer [Conway et al., 2012; Conway and Bromage, 2011] uses sparse bit arrays to represent the de Bruijn graph. Fermi [Li, 2012] uses modified versions of the algorithms described in this thesis to construct a string graph using the FM-index data structure.

1.3 Resequencing and Variant Calling

Often when we sequence an individual a reference genome for that species is already available. In the case of humans the individual is expected to match the reference at 99.9% of the bases. The *resequencing* problem is to discover the 1 in 1000 bases that differ between the individual and the reference. These differences can be *substitutions*, where the identity of a symbol is swapped with another symbol, or *indels*, where some bases have been inserted or deleted with respect to the reference. There are also larger *structural variants*. These are large deletions, copy number changes, inverted segments or chromosomal rearrangements. The process of finding the ways in which a sequenced genome differs from the reference is referred to as *variant calling*.

[1] Overlapping paired-end reads and at least one long-insert mate pair library

Standard approaches to variant calling will *map* the sequence reads to the reference genome. This is the process of finding the location on the reference genome that a sequence read (or read pair) was sampled from. The result of mapping is an *alignment* of a read against the reference, which is a one-to-one mapping from individual bases of the read to individual bases of the reference. The mapping and alignment process must take into consideration errors that occur during sequencing, for instance incorrectly identifying a base, and possible variants between the individual and the reference genome. Once all reads have been mapped to the reference, variants are called by finding consistent differences between the aligned bases and the reference genome. For example if the reference base is a T at a given position and all read bases aligned to that position are C, we may say there is a T to C substitution. The strength of the evidence for each variant will be typically assessed in a probabilistic model to distinguish between sequencing or alignment errors and true variants. A comprehensive description of the mathematics involved in the variant calling process can be found in [Li, 2011].

Variant calling by mapping reads to a reference genome is effective for isolated substitutions and small indels and has formed the basis of many resequencing projects, like the 1000 Genomes Project [1000 Genomes Project Consortium, 2010]. However, the mapping and alignment process can fail when there are significant differences between the individual and the reference genome, like multi-base substitutions or larger indels. In the worst case the sequence reads will be *misaligned* to the reference and false variant calls will be made. The primary source of misalignments (and therefore false positive variants) is polymorphic indels [Li and Homer, 2010]. For this reason, assembly-based methods of variant calling have recently been proposed [Iqbal et al., 2012; Li, 2012]. In Chapter 4, I will also address the variant calling problem with assembly-based algorithms.

1.4 Compressed Data Structures

A text *index* is a data structure which allows string queries to be performed without requiring scanning the full text. For example, given a text collection T and a pattern P, we may wish to check whether P is a substring of T, count

the number of occurrences of P in T or locate the positions of P in T. Text indices are used frequently in bioinformatics to accelerate searches over large sequence collections. Sequence alignment is a classic application of text indices. The BLAT [Kent, 2002] and SSAHA [Ning et al., 2001] algorithms use an index of short subsequences of a reference genome to find candidate mapping locations of a query sequence which are refined by dynamic programming.

Some of the most time efficient indices use suffix data structures. The suffix tree and suffix array data structures store an ordering of the suffixes of a text. The suffix tree can be searched for a pattern by matching the symbols of a pattern to labels on the edges of the tree. A suffix array uses considerably less space than a suffix tree and allows pattern matching via a binary search [Manber and Myers, 1990]. These data structures are extensively studied in Gusfield's classic sequence analysis text [Gusfield, 1997]. The suffix array is used widely in bioinformatics, for example in sequence clustering [Malde et al., 2003], repeat detection [Abouelhoda et al., 2002], microarray design [Li and Stormo, 2001] and k-mer counting [Kurtz et al., 2008].

Despite the widespread use of the suffix array, its memory requirements (8 bytes per input base for large sequence collections) has limited its use to relatively small analysis problems. In the past decade, *compressed* text indices related to the suffix array have been developed. These indices store a compressed version of the text, and define auxiliary data structures that allow the compressed text to be searched without the need for complete decompression. In 2000, Ferragina and Manzini described the FM-index (full-text, minute-space) [Ferragina and Manzini, 2000]. The FM-index is based on the Burrows-Wheeler transform [Burrows and Wheeler, 1994] and allows similar queries as the suffix array. Early applications of the FM-index in bioinformatics were for counting substrings in genomes [Healy et al., 2003] and finding local alignments between a query sequence and a reference genome [Lam et al., 2008]. The FM-index would later become the dominant data structure for mapping high throughput short reads to a reference genome [Langmead et al., 2009; Li and Durbin, 2009; Li et al., 2009]. The algorithms I develop in this thesis are based on the FM-index, and a full description of the data structures and how it is used is given in chapter 2. Compressed versions of the suffix array and suffix tree have been developed,

significantly shrinking their memory requirement [Grossi and Vitter, 2000]. As of yet, these structures have not been widely used within bioinformatics.

1.5 Overview of this work

In Chapter 2, I introduce my technical notation and definitions then describe an algorithm to efficiently construct the assembly string graph from a set of sequence reads, without the need to build the full overlap graph first. This will solve the time bottleneck for performing overlap-based assembly. The algorithm that I develop is based on the FM-index. As this is a compressed data structure, its memory footprint is much smaller than other full-text indices, like the suffix array. This will allow the reduction of the memory bottleneck of assembly. While the primary focus of this work is overlap-based assembly, in section 2.6 I will describe how the FM-index can be used as a memory efficient representation of a de Bruijn graph for all k.

In Chapter 3, I describe the implementation of these assembly algorithms into a fully functional sequence assembler called SGA. This assembler has algorithms for building the FM-index from very large sequence collections and correcting sequencing errors. I demonstrate the use of SGA on real sequence data from *Schizosaccharomyces pombe*, *Caenorhabditis elegans* and the human genome. Also, I compare the output of SGA to leading de Bruijn graph assemblers.

In chapter 4, I study the problem of finding differences between a pair of related genomes. I will address this problem by using assembly graphs to model the variation structure within the genomes. I will describe the use of both de Bruijn graphs and string graphs for this problem. Again, this work is based on the FM-index. This allows efficient access to the complete read sequences, which I will use when developing a probabilistic model to distinguish between sequence errors and true variation[1]. In Chapter 5, I study the sensitivity and accuracy of this approach with simulations. I also apply this approach to the problems of detecting newly acquired mutations in the offspring of two parents (*de novo* mutations), detecting somatically acquired mutations in cancer and detecting variation within a population of individuals.

[1]This work is performed in collaboration, details are provided at the start of Chapter 4

In Chapter 6, I offer concluding remarks including the prospects of expanding upon this work for upcoming sequencing technology.