IMPROVING GENOME ASSEMBLY BY IDENTIFYING RELIABLE SEQUENCING DATA

BY

RAJAT SHUVRO ROY

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Alexander Schliep

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2014

ABSTRACT OF THE DISSERTATION

Improving genome assembly by identifying reliable sequencing data

by

Rajat Shuvro Roy

Dissertation Director: Alexander Schliep

De novo Genome assembly and k-mer frequency counting are two of the classical problems of Bioinformatics. k-mer counting helps to identify genomic k-mers from sequenced reads which may then inform read correction or genome assembly. Genome assembly has two major subproblems: contig construction and scaffolding. A contig is a continuous sub-sequence of the genome assembled from sequencing reads. Scaffolding attempts to construct a linear sequence of contigs (with possible gaps in between) using paired reads (two reads whose distance on the genome is approximately known). In this thesis I will present a new computationally efficient tool for identifying frequent k-mers which are more likely to be genomic, and a set of linear inequalities which can improve scaffolding (which is known to be NP-hard) by identifying reliable paired reads.

Identifying reliable k-mers from Whole Genome Amplification (WGA) data is more challenging compared to multi-cell data due to the coverage variation introduced by the amplification step (MDA, MALBEC, etc.), which implies that applying a simple k-mer frequency cutoff is unreasonable. We observed that with sufficient coverage, using partial reads (read prefix of a certain length) of length approximately twice or less than that of the k-mer length recovers a large proportion of genomic k-mers while keeping

the proportion of erroneous k-mers low. We show that using partial reads for assembly and gene prediction recovers a significant proportion of genes and propose to use this approach for rapid pathogen detection in combination with Single Cell Genomics (SCG).

Thanks to SCG, it is now possible to isolate one single cell from environmental sample, extract its DNA and perform genetic sequencing without any need for culturing the cell in the lab. We show that current bioinformatic tools are capable of characterizing a novel organism by producing a draft genome assembly and gene annotation from single cell data of a MAST-4 stramenopile. This demonstrates the potential of SCG for genetic study of the vast majority of environmental organisms that has so far eluded scientists as they cannot be brought into culture, typically a necessity for future studies.

Acknowledgements

I thank my advisors, Prof. Alexander Schliep and Prof. Debashish Bhattacharya, Prof. Kevin Chen and Prof. Anirvan Sengupta for their guidance and support. They introduced me to some interesting computational problems in genetics and created opportunities for me to work and improve the state of the arts. I am very grateful to the Department of Computer Science, Rutgers University for supporting me. My lab members Pavel Mahmud, Ivani Lopez and John Weidenhoeft were instrumental in my achievements and I thank them for their contributions.

Over the course of my graduate study, I had the opportunity to come in contact and learn from many brilliant minds— faculty members, friends and fellow graduate students. I thank Prof. Muthu Muthukrishnan, Prof. William Steiger and Prof. Nina Fefferman for their inspiration and guidance during the early periods of my graduate life. I am grateful to my fellow graduate students Aleksandar Nikolov, Jason Perry, Mudassir Shabbir, Edinah Gnang, Chathra Hendahewa, Brian Thompson, Marie-Mathilde Perrineau and Moinul Zaber.

Lastly, I thank my parents Prem Chand Roy and Shubra Roy, my brother Pinak Shuvro Roy and my wife Tanni Chakraborty for always believing in me.

Dedication

To my parents: Prem Chand Roy and Shubra Roy.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Genome sequencing has revolutionized many fields of biological research. In 1953, James Watson and Francis Crick presented the structure of the DNA-helix (WATSON and CRICK, 1953), the molecule that carries genetic information from one generation to the other. For this, they were awarded the Noble prize in physiology in 1962. A few years earlier, in 1944, Oswald Avery et al. (Avery *et al.*, 1944) demonstrated that genes were characterized by nucleic acids. This paved the way to the discovery of the genetic code (how a series of three nucleotide acids encode a protein) (Crick *et al.*, 1961) that eventually enabled us to characterize genes as a sequence of nucleotides that encode proteins. Naturally, this led to the realization that the genotypic differences between individuals and organisms can be attributed to their genetic differences. Researchers realized the potential of determining the genomic (DNA) sequence of organisms and individuals in order to get more insight into evolution, phylogeny, or diseases like cancer and research into determining the genomic sequences of organisms began.

1.1 Genome sequencing and assembly

Genome sequencing is the process of experimentally determining the Deoxyribonucleic acid (DNA) sequence which can be thought of as long string made up of four characters 'A', 'C', 'G' and 'T'. Unfortunately current technology only produces very short subsequences (called reads) of the genome. The goal is to assemble the reads into longer contiguous sequences (or contigs) so that important genomic entities like protein coding genes may be identified. The process of assembly is complicated by the fact that the reads are short, they may contain errors (base substitutions, insertions

or deletions) and that the genome can be repetitive in structure. Assembly is generally divided into contig building, scaffolding and finishing, each step trying to improve on the previous stage to produce longer sequences until we have the complete genome or set of chromosomes. This gives rise to several interesting computational problems like reconstructing a (super) string from (erroneous) short substrings, error correction of substrings, relative ordering and orientation of contigs into scaffolds, etc. For small sized genomes that are not very repetitive, like those of viruses or bacteria, considerable progress have been made over the last few decades. However, as we move on to more challenging cases like mammalian and plant genomes which are longer and have a more complicated repeat structure, meta-genomes (a set of genomes sequenced together) or single-cell libraries where the sequencing shows extreme coverage variation, the computational problems get harder and often extremely resource intensive. Therefore, the community has a great demand for better computational solutions and my research is focused on developing efficient algorithms for some of these classical bioinformatic problems.

1.2 Characterizing organisms from their genomes

The genome contains hereditary information in the form of protein coding genes. As proteins are the building blocks of the body, identifying genes can provide a lot of information about the capabilities of an organism and is therefore of great importance to life scientists. Since the genome encodes the genes, reconstructing the genomic sequence (by genome sequencing and assembly) is so important to the community. Moreover, as the genome is the carrier of hereditary information, the genomic sequence can also inform us about phylogeny, the study of evolutionary relationships between organisms. Genetics is also playing a key role in understanding important evolutionary mechanisms like Horizontal gene transfer (HGT), medical phenomenon like development of cancer or drug-resistance.

Scientists have very successfully performed genetic analysis for multi-cell organisms and unicellular microbes that can be cultured in labs. But the vast majority of the Earth's biodiversity is in microbes which cannot be brought into culture (Pawlowski

et al., 2012; Guiry, 2012) and for these we have to resort to either Metagenomics (MG) or SCG. Metagenomics also informs about the biodiversity and collaborative metabolism of a microbial community. Both these areas have the potential to make huge impacts in medicine and life sciences but currently face significant challenges in genome assembly. Some of these challenges may be overcome by improved sequencing methods like reduced coverage bias for WGA libraries, which is an active area of research (Zong *et al.*, 2012; Stepanauskas, 2012a), while others (like reducing chimeric contigs in meta-genomic assembly) are computational in nature. Thus, the computational and sequencing methods are co-evolving into a powerful tool for environmental biology.

1.3 Rapid pathogen detection

For the medical community, quick and accurate diagnosis is a major concern for appropriate therapy. Culture-based methods have dominated pathogen detection for more than a century and the adoption of modern methods have been slow (van Belkum *et al.*, 2013). Nucleic acid detection based methods, which depend on detecting genomic sequences that serve as identifiers, have recently replaced some traditional methods but identification of drug-resistance is still lagging behind (Ecker *et al.*, 2010). Whole Genome Sequencing (WGS) is expected to provide a better alternative to understanding drug-resistance by producing a draft genome assembly which then allows you to make *in silico* protein predictions. The challenge is to avoid the time-consuming step of culturing, which is possible with SCG but introduces sequence analysis challenges in the form of amplification bias. If we can overcome these challenges, WGS can be a very effective, accurate and fast diagnostic tool.

1.4 Thesis overview

Counting k-mer frequencies is one of the fundamental problems of bioinformatics and have been used to inform error correction (Kelley *et al.*, 2010; Yang *et al.*, 2010; Medvedev *et al.*, 2011), *de novo* assembly (Yang *et al.*, 2013), etc. For large read

libraries, computing k-mer frequencies can be a major computational challenge. In Chapter 3, we present a tool named Turtle (Roy *et al.*, 2014b) that is capable of counting frequencies of frequent k-mers from a 44x Human library in approximately 110 minutes using 109 GB of space. Our method improves running time by using cache-efficient algorithms. Our space efficiency is due to the use of compact data structures like Bloom filters and contiguously allocated fixed sized arrays.

In Chapter 4, we present a set of linear equations capable of filtering out unreliable paired-reads and also predicting the relative position and orientation of contigs connected with paired-reads. This greatly simplifies the scaffolding problem and we show (Roy *et al.*, 2012) that a very simple scaffolding algorithm is capable of outperforming state of the art scaffolders. The equations are derived from the linear geometry of contigs on the genome that dictates that two contigs cannot overlap by a large number of bases. Our experiments also show that correct contigs produce a much cleaner contig graph and thereby further simplify the scaffolding problem instances.

In Chapter 5, we present a study where we use SCG to generate the first draft genome assembly from a cell belonging to the broadly distributed group of MAST-4 marine stramenopiles (Roy *et al.*, 2014a). Since these cells cannot be cultured, we used WGA methods for whole genome sequencing and performed sequence analysis using a set of in-house and third party tools. We performed genome assembly with specialized single-cell assemblers and then made gene predictions to identify ca. 7,000 protein-encoding genes in the MAST-4 genome. Using the identified protein-encoding genes, we were able to robustly position the marine organism into the Tree of life (ToL) using multigene phylogenetics and thus gain insights into its metabolic capabilities. This project not only produced the first draft genome for the MAST-4 family, but also demonstrated the feasibility of producing a reasonable characterization of small eukaryotes using available bioinformatic tools and sequencing methods.

SCG can be very effective against rapid pathogen detection as it allows us to circumvent the time consuming step of culturing. However, assembly and analysis of single cell libraries is complicated due to the high coverage bias introduced by the WGA process required to produce enough DNA material for sequencing. This results in high rate

of erroneous k-mers which makes sequence analysis like assembly difficult. In Chapter 6, I present our ongoing work that shows that using partial reads (a prefix of a fixed length of the read), we can reduce the proportion of erroneous k-mers while still producing reasonable assembly and gene prediction which may be used for rapid pathogen detection and understanding its drug resistance.

Chapter 2

Basic concepts

In this chapter, I will introduce some basic concepts of sequence analysis and efficient computation. These concepts will be frequently used in later chapters where I present my contributions in more details.

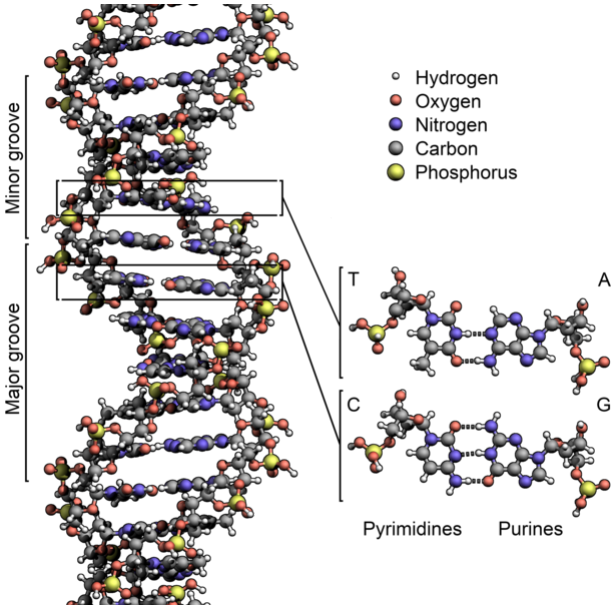
2.1 The Genome

The genome is the carrier of the complete hereditary information of an organism. In most organisms, it is encoded by DNA, and in some viruses, by RNA (Ribonucleic acid). From a computational perspective, the DNA molecule is a string of four characters A, C, G, T, representing the four distinct unit molecules Adenine, Cytosine, Thymine and Guanine (called bases). The DNA comes as a pair of strands (Fig 2.1) where bases from opposite strand couple with each other—Adenine with Thymine and Cytosine with Guanine. Because of this coupling, one strand of the DNA can be used to unambiguously determine the other. These strands are said to be reverse complements of each other. The genome may be present as a single sequence (e.g. in most prokaryotes) or in multiple pieces called chromosomes. Depending on the number of copies of each chromosome present, an organism may be categorized as haploid (e.g. *E. coli*), diploid (e.g. humans) or polyploid (e.g. corn).

2.2 Genome sequencing

The process of experimentally determining the DNA sequence is known as genome sequencing. Unfortunately, current technologies are incapable of reading complete DNA sequences. Depending on the technology used, we can reliably (with small error rate) read only a few hundred base pair from the ends of a DNA fragment. Therefore, the

Figure 2.1: The structure of DNA with details of the structure of the four bases, adenine, cytosine, guanine and thymine, and the location of the major and minor groove (source `http://en.wikipedia.org/wiki/DNA`).



genome is first sheared into small fragments and the ends of the fragments are sequenced to produce small sub-sequences of the DNA string known as reads.

Though the structure of the DNA was known since 1953, a few decades were required to establish reliable methods of reading the DNA molecule. The 1970's saw several notable developments in the field of sequencing, the most notable one being that by Frederick Sanger—"DNA sequencing with chain-terminating inhibitors" (Sanger *et al.*, 1977), also commonly known as Sanger sequencing, earning him the Noble prize in chemistry in 1980. Since its discovery, Sanger sequencing has been the sequencing method of choice for large scale genome projects like the International Human Genome Consortium (Lander *et al.*, 2001), The Human genome project (Venter *et al.*, 2001), Mouse Genome Sequencing Consortium(Consortium *et al.*, 2002), etc., till the recent development of Next Generation Sequencing (NGS) or High Throughput Sequencing (HTS) in the mid-2000's. Though the quality and efficiency of Sanger sequencing improved over the years, the cost of sequencing (see Table 2.1) was too high for routine sequencing in most laboratories. Some of the NGS technologies are briefly compared in Table 2.1. Note that, the NGS platforms provide much shorter and more erroneous

Table 2.1: Some common Next Generation Sequencing technologies, compared to first generation (Sanger) sequencing. (Sources: Hoff (2009), Gilles *et al.* (2011), Quail *et al.* (2012))

Generation	Company	Platform	Read length	error rate (%)	cost per kb
First/Sanger	Life Technologies	3730xl	600 - 1000	0.001-1	$10.00
Next	Roche/454	GS Junior	400	0.02-50	$0.031
Next	Illumina	MiSeq	36-250	0.8	$0.0005
Next	Ion torrent	PGM	200	1.71	$0.001
Next	Pac Bio	RS	1500	12.86	$0.002

reads compared to Sanger sequencing but are considerably cheaper. This cost effectiveness have enabled genome sequencing to become a routine procedure, but the length of the reads, amount of data and their error rate demanded new algorithms for their analysis. We will briefly describe such algorithms in Section 2.3. Recently, NGS alone have been used to produce assemblies for mammalian genomes like Giant panda (Li *et al.*, 2010c) and Chinese human (Li *et al.*, 2010a). In some cases, e.g. Gorilla genome (Scally *et al.*, 2012), researchers are using a combination of Sanger (long) and Illumina (short) sequencing data to produce assemblies. Such combinations are very helpful for assembling repeat rich genomes like those of mammals and plants (English *et al.*, 2012). Hopefully, such combination will be more routine with improvements in Next generation long read platforms like *Pac Bio* which currently have a very high error rate (Table 2.1).

2.2.1 Paired sequencing

To overcome the limitations of short length, paired read sequencing was introduced, where reads are sequenced in pairs. Ideally, reads of a pair should have a fixed distance on the genome but in reality, this distance between follows a distribution. Paired read sequencing comes in two flavors: paired-end and mate-pairs. Paired-end involves reading from both ends of a DNA fragment and is mostly done in Illumina platform. The DNA fragment size (or insert length) is limited to < 1000 (Donmez, 2012).

Mate-pair sequencing allows a much larger insert length ($\leq$ 20kbp) (Donmez, 2012)

and is not limited to Illumina platform only, but the tedious procedure of library preparation and the probability of chimeric pairing (two separate DNA fragments mistakenly concatenated and sequenced to produce a false pair) limits the use of this method.

It must be mentioned that the insert length distribution may be very difficult to control and varies from one library preparation technique to another. The uncertainty of the insert length must be carefully taken into consideration for all analysis involving paired reads.

2.3 Genome Assembly

Current sequencing technologies allow reliable reading of a DNA fragment that is only a few hundred bases in length (Table 2.1). Unfortunately, this is far shorter than bacterial genomes (usually a few Mega base pairs in length) and extremely short for mammalian genomes (usually Giga base pairs in length). Therefore, the complete genome has to be assembled from much shorter sequenced reads using computational methods. This is known as genome assembly.

2.3.1 Challenges

Note that, the sequenced DNA reads are assumed to originate from uniform random positions of the genome. In an effort to ensure that every position of the genome is covered by at least a few reads, some over-sampling is performed. This over-sampling is quantified in terms of "coverage", the ratio of the total number of base pairs sequenced to the size of the genome. Higher coverage is desirable for completeness of assembly, but this also increases the amount of sequencing data, which, in the presence of errors, complicates the assembly process. The assumption of uniformity of coverage is also not correct: it is known that certain parts of the genome are more difficult to sequence than others (Benjamini and Speed, 2012).

Apart from the technical difficulty of sequencing, the structure of the genome under consideration also introduces additional challenges—prokaryotic genomes usually contain less repeated structures compared to eukaryotic genomes (e.g. 3.4% for yeast vs.

44% for humans) (Brown, 2002), some eukaryotic genomes (e.g. mammals and plants) have multiple copies of their chromosomes (termed ploidity), etc. Over the decades, a few major approaches to genome assembly have emerged. We provide a brief overview of these approaches in the next subsection.

2.3.2 Assembly approaches

Early assembly tools like the Staden package (Staden, 1979) were designed to assist humans to identify and merge overlapping sequence reads. As the size of the dataset grew to billions of reads, fully automated computational tools were developed for assembly (called assemblers). Early tools followed a greedy approach— find overlaps between each pair of reads, merge the reads sharing the best overlap (provided it is more than a required minimum) and iterate the procedure till reads can be merged (Simpson, 2012). Special care were needed to handle merging reads coming from similar repeats. Moreover, it was difficult to formally argue anything about the quality of the assembly.

In an effort to formalize the genome assembly problem, Kececioglu and Myers formulated it as the Shortest Common Superstring problem (Kececioglu and Myers, 1993). They represented the reads and the overlap between them in an overlap graph where each read was a node and two nodes are connected by an edge if the corresponding reads had significant overlap. Then, the assembly problem can be stated as finding a walk through each node (the Hamiltonian path problem). Since overlap computation can take $O(n^2)$ where n is the total number of bases in the read library and computing Hamiltonian path is NP-Complete, computationally, this method was not a very attractive option. In practice however, the overlap computation stage was found to be the major bottleneck. Various methods for improving the overlap computation have been proposed, the most notable one being that by Simpson and Durbin (Simpson and Durbin, 2012) that brought the computational complexity down to $O(n)$. Collectively, this strategy of genome assembly is known as the Overlap Layout Consensus (OLC) assembly.

In 2001, Pavel Pevzner proposed an alternative formulation known as the *de Bruijn* graph approach (Pevzner *et al.*, 2001) of assembly where the reads are initially broken

into k-mers (subsequences of length k) and a graph is built using exact overlap between $(k-1)$-mers: each k-mer represents an edge connecting two $k-1$-mers produced by taking the first and last $(k-1)$ bases of that k-mer. In the graph, Eulerian paths are determined and these paths spell out the underlying genomic sequence. The simplicity of the graph structure, the efficiency of Eulerian path computation (computable in linear time), and the success with which it handled repetitive sequences made this approach very popular for high throughput, short sequence reads like NGS data.

2.3.3 Error correction

Unfortunately, sequencing data comes with errors (substitutions, insertions or deletions) and such errors makes the computational process of genome assembly harder and more resource intensive. For example, in a *de Bruijn* graph approach, the amount of space required is proportional to the number of distinct k-mers in the read library. A single base call error in a read can introduce up to k erroneous k-mers. Thus, erroneous k-mers may out-number correct k-mers in the read library (Roy *et al.*, 2014b). While assembling large genomes such as mammals or plants, this can lead to computational in-feasibility due to very high memory requirements. Therefore, researchers have proposed error correction as a preprocessing step for large assembly problems. It has also been claimed that error correction improves assembly quality (Yang *et al.*, 2013) and therefore, error-correction is now recommended to be a standard procedure for large genome assembly projects (e.g. Li *et al.* (2010c,a)).

2.3.4 Scaffolding

The assembly procedures described above produces 'contigs' (a contiguous subsequence of the genomic sequence). Usually, the assembly is broken into contigs due to lack of coverage or the presence of repeats. Using paired reads, contigs may be oriented and arranged in a linear order (with possible gaps in between) called a scaffold such that the orientation and order reflects their true orientation and order in the genome. The most popular scaffolding strategy is to construct a contig graph in which nodes represent contigs and edges represent sets of paired-reads connecting two contigs (i.e.,

the two reads of the mate pair fall in the two different contigs). The edges are given weights equal to the number of paired-reads connecting the two contigs. Ideally, we would like to find an orientation and order assignment of the contigs such that the minimum number of paired-reads are violated. A paired-reads is violated when the contigs it is connecting do not have the relative orientation or position suggested by the paired-reads.

Just finding the optimal orientation assignment is reducible to the Maximum weight Cut problem, which is known to be NP-complete. Consequently, finding the optimal walk to get the optimal scaffolding is also NP-complete. The genome can (and often does) have repeated regions; e.g., approximately 44% of human genome is accounted for by repeats (Brown, 2002). Ideally, the contig graphs should have a linear structure, but due to repetitive structures of the genome, loops and cycles are introduced; making the problem more difficult to solve. Scaffolding is an active area of research which has produced several scaffolders in recent years (Salmela *et al.*, 2011; Koren *et al.*, 2011; Dayarian *et al.*, 2010; Donmez and Brudno, 2013).

2.4 Characterizing an organism from assembled genome

Once a draft genome assembly is available, there are several ways to learn more about the organism. *In silico* gene prediction tools can predict genes from the assembly which can enable phylogenetic analysis. Using homologs of the predicted proteins, we can also learn about the metabolic capabilities of the organism. In the following subsections, we briefly discuss currently available methods.

2.4.1 Gene prediction

Gene prediction involves identifying regions of the genome that encodes genes. In earlier days, gene finding was mostly performed by wet-lab experiments on living cells and organisms. But now, with the advances in genome sequencing, most of gene predictions are performed using computational tools. However, confirming a predicted gene still demands an *in vivo* process with gene knockouts and other assays (Sleator, 2010).

Computational gene prediction tools are largely categorized into two classes: *ab initio* tools that use known genes to learn gene structures and make further predictions and *similarity-based* tools that use sequence homology to known proteins (Stanke and Waack, 2003). In the presence of a similar gene, similarity-based methods often outperform *ab initio* gene predictors but obviously, when such similar genes are not available, the *ab initio* methods present a more general approach (Stanke and Waack, 2003).

Similarity-based tools rely on sequence alignment for finding genes in the assembled genome. Once they find sufficient similarity between a genomic region and a known Expressed sequence tag (EST), Protein or Gene encoding region of the genome, this similarity is used to infer gene structure and function of the gene. Obviously, this method is limited by the availability of known EST, proteins or gene encoding genomic regions.

Ab initio gene prediction tools are more powerful because they use gene structure to train parameters in their models of the biological signals and models for coding and non-coding regions. They rely on two types of sequence information: signal sensors (e.g. splice sites, branch points, start and stop codons, etc.) and content sensors (codon usage) (Wang *et al.*, 2004). The Hidden Markov Model (HMM) based tools like Augustus (Stanke and Waack, 2003), GENSCAN (Burge and Karlin, 1997), etc. have proved to be the most successful. The major limitations for these methods is the lack of knowledge of gene structure for novel organisms. Fortunately, some proteins are expected to be present in all organisms (also known as core proteins) of a domain and these core proteins can be used to learn gene structures of novel organisms. CEGMA (Parra *et al.*, 2007) is an alignment based specialized gene prediction tool that identifies the core proteins in an assembly and combined with an *ab initio* gene prediction tool can provide a mechanism for gene prediction for novel organisms.

2.4.2 Phylogenetic analysis

Members of the same species have largely similar physiological capabilities. Closely related species have somewhat similar and distantly related organisms are expected to be very different in terms of physiological capabilities. Since the genome is the carrier of

hereditary information, in terms of genetics, this means that closely related organisms have similar gene inventory while distantly related ones are expected to have different gene sets. Therefore, sequence similarity of proteins or the gene encoding regions of the genome can provide a measure of phylogenetic distances between both inter and intra species organisms. This measure of distance may be used to build a phylogenetic tree of life (ToL). Various paradimgs of phylogenetic tree constructions exist. Some of the widely used ones are neighbor-joining, UPGMA, maximum parsimony and maximum likelihood (Rizzo and Rouchka, 2007). Constructing the optimal tree using many of these techniques is NP-hard (Roch, 2006) and therefore, heuristic search and optimization methods are used in combination with tree-scoring functions to identify a reasonably good tree that fits the data.

Although this idea is very intuitive, applying it to distantly related organisms can be challenging due to the following reasons. Distantly related organisms may have very different gene sets and therefore selecting an appropriate subset of (common) phylogenetically stable genes for comparison is difficult but critical to accurate measurements (Capella-Gutierrez *et al.*, 2014). The ribosomal genes (16s/18s) have been widely used as phylogenetic markers due to their ubiquity, ease of amplification and appropriate level of conservation for most purposes (Capella-Gutierrez *et al.*, 2014). Many other phyogenetic markers have been introduced, some of them for a specific taxa or only suitable for specific taxonomic levels. Previous studies have shown that different genes may be better suited to resolve different parts of the phylogeny (Townsend, 2007), and hence it is important to consider the resolving power of combinations of marker genes. Ideally, we would like to have the minimal set of genes that has sufficient information to reconstruct a phylogeny. In the previous subsection, we introduced the core proteins which are expected to be present in all organisms of a domain. Our work presented in Chapter 5 show that in the absence of any specific information about the phylogenetic relation between a set of organisms, the core proteins are well suited for phylogenetic study.

2.4.3 Understanding metabolic capabilities

A metabolic pathway is a series of chemical reactions occurring within a cell. It involves step-by-step modification of initial molecules to products needed for the cell. Generally speaking, the biological function of the living cell is a result of many interacting molecules and cannot be attributed to just a single gene or a single molecule (Kanehisa and Goto, 2000). KEGG (Kanehisa and Goto, 2000) attempts to link a set of genes with a network of interacting molecules in the cell, such as a metabolic pathway or a complex, representing a higher order biological function. By knowing the genes an organism possesses, it is possible to predict its higher order capabilities like photosynthesis, the existence of urea or Kreb cycle, etc. Thus we can predict an organism's physiological capabilities and if desired, can design targeted experiments for validating them too.

2.5 Single Cell Genomics (SCG)

Genome assembly can be a very powerful tool for scientific discovery. Certain parts of the genome, known as genes, encode proteins in the form of amino acid sequences. Proteins perform important cellular functions in organisms (Lodish *et al.*, 2004) and can provide valuable information about evolutionary relationship between organisms, an organism's capabilities or its metabolic functions. Traditionally, an organism would be cultured in laboratory and its mRNA would be sequenced and assembled (also known as Transcriptome assembly) to identify candidate proteins for further analysis like mass spectrometry-based identification (Evans *et al.*, 2012). A broad swath of microbial biodiversity cannot be cultivated in the lab and is therefore inaccessible to such conventional approaches. Researchers are interested to gain knowledge about such organisms to elucidate their genome evolution, their places in the ToL and their metabolic capacities. SCG is a recent promising approach whereby an individual cell is captured from nature and genome sequencing data are produced from the amplified single cell DNA. This data may be assembled and genes may be predicted *in silico*, which in turn, opens up the possibility of genome-wide study of the organism.

SCG may also be instrumental for rapid pathogen detection. Traditionally, identification of a pathogen involved isolation, culture and pathological testing. SCG can eliminate the time consuming step of culturing and we envision a pipeline where the pathological testing is replaced with parallel sequencing and bioinformatic analysis. For conditions like sepsis, where the speed of diagnosis is crucial for reducing mortality, having this faster pipeline can be a significant clinical advantage. However, there are significant technical hurdles that needs to be overcome before this becomes a reality as the sequence analysis becomes significantly complicated due to the coverage bias introduced by the WGA step that is required to produce enough genetic material for sequencing.

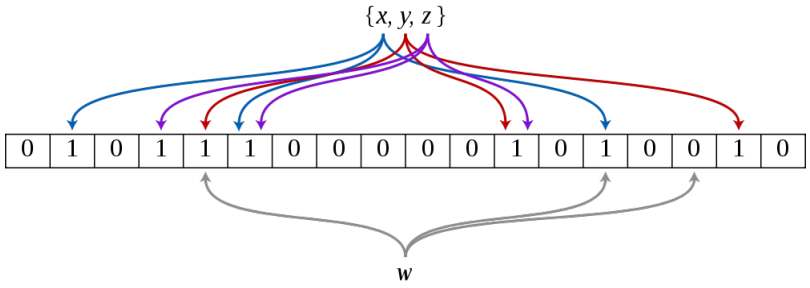
2.6 Efficient data structures and algorithms

NGS had made genome sequencing inexpensive and has led to a data generation rate that has overtaken Moore's law (Sboner *et al.*, 2011). This implies that while we will be able to generate more and more sequence bases at a fixed cost, we will soon lack the facilities to store, process, analyze and maintain the data generated. Therefore, design and implementation of efficient bioinformatic tools is vital for continuing sequence based analysis. Broadly speaking, computational efficiency involves two factors: space (memory) and time. In this section, we introduce a space efficient data structure and some basic concepts of how to achieve time efficiency by taking advantage of locality of memory access.

2.6.1 Bloom filter

Suppose we are observing a stream of items and at any point, want to be able to answer whether a given item has already been seen. We may maintaining the set of observed items in a hash table or a binary search tree or any other efficient data structure which enables efficient item lookups. However, this require explicit storage of the items. For a large set of items, this can be very memory consuming. If the set membership query can tolerate a small amount of false positive (an item that is not a member of the set

Figure 2.2: An example of a Bloom filter, representing the set x, y, z . The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set x, y, z , because it hashes to one bit-array position containing 0. For this figure, $L = 18$ and $k = 3$. Source:`http://en.wikipedia.org/wiki/Bloom_filter`



is declared be be so), a much more memory efficient probabilistic data structure known as the Bloom filter (BF) may be used.

A Bloom filter is a space efficient probabilistic data structure which can answer set membership queries with some prescribed, small false positive rate. A Bloom filter works as follows: A large bit-array(B) of size L is initialized to 0. Given an item x, k hash values $(h_1, h_2, \ldots, h_k)$ using k independent hash functions (within the range $[0, (L-1)]$) are computed. To insert the item into the Bloom filter, we set all of the bits $B[h_1], \ldots, B[h_k]$ to ones. To check if x is in the Bloom filter, we check all the bits $B[h_1], \ldots, B[h_k]$, and if they are all set to one, with high probability, this item is in the Bloom filter. If not, it is certainly not in the Bloom filter. The reason we cannot be absolutely sure about the presence of the item in the Bloom filter is because just by chance, all of the bits $B[h_1], \ldots, B[h_k]$ might have been set by other items while being inserted into the Bloom filter. Each item occupies k bits in the Bloom filter and usually k is much smaller than the item itself. More interestingly, k is independent of the size of the item. Under the assumption that the hash functions select a bit position uniformly at random, the false positive rate of the Bloom filter is given by $(1 - e^{kn/L})^k$(`http://en.wikipedia.org/wiki/Bloom_filter`), where n is the number of items inserted into the Bloom filter. This means that the false positive rate may be reduced by increasing the size of the Bloom filter. The optimal number of hash

functions k is approximated by $L/n \ln 2$.

Cache-efficient Bloom filter

Note that, by design, the bit locations for an item are randomly distributed and this results in better false positive rate. When the size of the Bloom filter is large, each bit inspection and update is likely to incur one cache miss. So, the total number of cache miss per item would be k. On the contrary, if the bit-locations are localized to a few consecutive bytes (a block), each item lookup/update will have a small number of cache misses. This can be done by restricting $h_1, \ldots, h_k$ to the range $[h_1(x), h_1(x)+b]$ where b is a small integer. The bit pattern for each item can also be precomputed. This is called the Pattern-blocked Bloom Filter. Putze *et al.* (2010) observed that the increase in false positive rate due to this localization and precomputed patterns can be countered by increasing L by a few percent. Cache efficiency is a major issue for fast computation since a cache miss is more than a hundred times slower than a cache hit (Levinthal, 2008). Therefore, using cache efficient Bloom filters is highly recommended for high performance computation.

2.6.2 Cache-efficient algorithms

Cache efficiency has recently been identified as a major obstacle to achieving high performance on modern architectures, motivating the development of cache-oblivious algorithms and data structures (Bender *et al.*, 2005) which optimize the cache behavior without relying on information of cache layout and sizes. This becomes more important when the dataset is large as it may be divided among various levels of memory like Cache, RAM and Disk. Algorithm efficiency is traditionally reported in terms of asymptotic running times like $O(n), O(n \log n), O(n^2), \ldots$, etc., and usually a smaller asymptotic complexity translates into faster algorithms in practice. But if two algorithms differ in complexity by only a small factor (e.g. $O(n)$ vs. $O(n \log n)$), and the asymptotically slower one is much more cache efficient, in practice, the asymptotically slower algorithm may outperform the asymptotically faster one. In Chapter 3, we report such a case and highlight the importance of cache-efficiency in designing high

performance algorithms for Bioinformatics and Big data analysis.

Chapter 3

Frequent k-mer counting

Counting the frequencies of k-mers in read libraries is often a first step in the analysis of high-throughput sequencing data. K-mers play an important role in many methods in bioinformatics because they are at the core of the *de Bruijn* graph structure (Pevzner *et al.* (2001)) that underlies many of today's popular *de novo* assemblers (Simpson *et al.* (2009); Zerbino and Birney (2008a)). They are also used in assemblers based on the overlap-layout-consensus paradigm like Celera (Miller *et al.* (2008)) and Arachne (Jaffe *et al.* (2003)) as seeds to find overlap between reads. Several read correction tools (Kelley *et al.* (2010); Liu *et al.* (2012); Medvedev *et al.* (2011)) use k-mer frequencies for error correction. Their main motivation for counting k-mers is to filter out or correct sequencing errors by relying on k-mers which appear multiple times and can thus be assumed to reflect the true sequence of the donor genome. In contrast, k-mers which only appear once are assumed to contain sequencing errors. The frequent k-mers constitute a reduced but error-free representation of the experiment, which can inform read error correction or serve as the input to de novo assembly methods.

We present a novel method that balances time, space and accuracy requirements to efficiently extract frequent k-mers for high coverage libraries and large genomes such as human. Our method is designed to minimize cache-misses in a cache-efficient manner by using a Pattern-blocked Bloom filter to remove infrequent k-mers from consideration in combination with a novel sort-and-compact scheme (instead of a Hash) for the actual frequency counting. Whereas this increases theoretical complexity, the savings in cache misses reduce the empirical running times. A comparison to the state-of-the-art shows reduced memory requirements and running times.

3.1 Publication Note

The work described in this chapter was previously published in Roy *et al.* (2014b). The work described is the sole work of the author, under the supervision of his PhD supervisors.

3.2 Related work

In a genome of size g, we expect up to g unique k-mers. This number can be smaller due to repeated regions (which produce the same k-mers) and small k, since smaller k-mers are less likely to be unique, but is usually close to g for reasonable values of k. However, depending on the amount of sequencing errors, the total number of k-mers in the read library can be substantially larger than g. For example, in the DM dataset (Table 4.1), the total number of 31-mers is approximately 289.20M whereas the number of 31-mers occurring at least twice is approximately 131.82M. The size of the genome is 122 Mbp (Mega base pairs). This is not surprising because one base call error in a read can introduce up to k false k-mers. Consequently counting the frequency of all k-mers, as done by Jellyfish (Marcais and Kingsford (2011)), which is limited to $k \leq 31$, requires $O(N)$ space where N is the number of k-mers in the read library. This makes the problem of k-mer frequency counting very time and memory intensive for large read libraries like Human. We encounter similar problems for large libraries while using Khmer (Pell *et al.*, 2012) which uses a Bloom filter (Bloom, 1970) based approach for counting frequencies of all k-mers. Ideally, the frequent k-mer identifier should use $O(n)$ space where n is the number of frequent k-mers ($n \ll N$). The approach taken by BFCounter (Melsted and Pritchard, 2011) achieves something very close to this optimum by ignoring the infrequent k-mers with a Bloom filter and explicitly storing only frequent k-mers. This makes BFCounter much more memory efficient when compared to Jellyfish. However, the running time of BFCounter is very large for two reasons. First, it is not multi-threaded. And second, both the Bloom filter and the Hash table used for counting incur frequent cache misses. Additionally, BFCounter is also limited to a count range of 0-255 which will often be exceeded in single-cell

experiments due to the large local coverage produced by whole genome amplification artifacts. A different approach is taken by DSK (Rizk *et al.*, 2013) to improve memory efficiency. DSK makes many passes over the reads file and uses temporary disk space to trade off the memory requirement. Though Rizk *et al.* (2013) claimed DSK to be faster than BFCounter, on our machine using an 18TB Raid-6 storage system, DSK required more wall-clock time compared to BFCounter. Therefore, we consider DSK without dedicated high-performance disks, e.g. solid state, and BFCounter to be too slow for practical use on large datasets. A disk based sorting and compaction approach is taken by KMC (Deorowicz *et al.*, 2013), which was published while the manuscript Roy *et al.* (2014b) was in preparation, and it is capable of counting k-mers of large read libraries with limited amount of memory. However, in our test environment, we found it to be slower than the method described here.

In this chapter, we present Turtle, our frequent k-mer counting tools that achieves space efficiency by filtering out infrequent k-mers with a Pattern Blocked Bloom filter and time efficiency by using cache-efficient sorting and compaction based frequency counting algorithm. Turtle comes in two flavors: *scTurtle* and *cTurtle*. *scTurte* reports k-mers with their frequency counts while *cTurtle* only reports the frequent k-mers. By avoiding frequency counting, cTurtle is able to work with much larger datasets compared to scTurtle.

3.3 scTurtle

By a k-mer, we always refer to a k-mer and/or its reverse complement. Our objective is to separate the frequent k-mers from the infrequent ones and count the frequencies of the frequent k-mers. To achieve this, first we use a Bloom filter to identify the k-mers that were seen at least twice (with a small false positive rate). To count the frequency of these k-mers, we use an array of items containing a k-mer and its count. These are the two main components of our tool. Once the counts are computed, we can output the k-mers having frequency greater than the chosen cutoff. For the sake of cache efficiency, the Bloom filter is implemented as a Pattern-blocked Bloom Filter (Putze *et al.*, 2010). It localizes the bits set for an item to a few consecutive bytes (block)

and thus reduces cache misses. The basic idea is as follows: when a k-mer is seen, the Bloom Filter is checked to decide if it has been seen before. If that is the case, we store the k-mer in the array with a count of 1. When the number of items in the array cross a threshold, it is sorted in-place, and a linear pass is made, compressing items with the same k-mer (which lie in consecutive positions of the sorted array) to one item. The counts add up to reflect the total number of times a k-mer was seen. Note that, this strategy is very similar to computing a run-length encoding (Salomon, 1997) the items. Our benchmarking (Table 3.1) shows that this very simple approach of storing items and their frequencies is faster than a Hash-table based implementation. An outline of the algorithm is given in Algorithm 1. More details are provided in the following subsections.

Note that the improved efficiency of sort and compaction also suggests that it can speed up the k-mer counting step for all *de Bruijn* graph based assemblers where k-mer counting is required for building the graph. We found that ABySS (Simpson *et al.*, 2009) and SPAdes (Bankevich *et al.*, 2012) requires 3,660 and 2,144 seconds respectively for k-mer counting on the DM dataset (see Table 4.1). But the sort and compaction method takes only 523.41 seconds. We provide a single threaded preliminary tool that implements this method for counting all k-mers and their frequencies called *aTurtle*.

Algorithm 1 scTurtle outline

Let, S be the stream of k-mers coming from the read library, BF be the Bloom filter, A be the array to store k-mers with counts, t be the threshold when we apply sorting and compaction.
for all k-mer$\in S$ **do**
  if k-mer present in BF **then**
    Add k-mer to A
  end if
  if $|A| \geq t$ **then**
    Apply sorting and compaction on A
  end if
end for
Apply sorting and compaction on A.
Report all k-mers in A with their counts as frequent k-mers and their counts.

Table 3.1: Comparison of Sort and Compress and Hash table based implementations for counting items and their frequencies. Jellyfish is a highly optimized hash table based implementation for the k-mer counting problem. We also compare against general purpose tools that uses Google sparse/dense Hash maps for storing k-mers and their counts.

Method	**Number of k-mers**			
	458M		2.2B	
	Time [sec]	**Space [GB]**	**Time [sec]**	**Space [GB]**
Sorting and compaction	**153.37**	2.70	**523.41**	**7.10**
Jellyfish	296.49	**2.40**	1131.70	7.20
Google Dense Hash	626.77	20.47	6187.95	40.38
Google Sparse Hash	1808.48	7.44	28069.18	10.60

3.3.1 k-mer extraction and bit-encoding

For space efficiency, k-mers are stored in bit-encoded form where 2-bits represent a nucleotide. This is possible because k-mers are extracted out of reads by splitting them on 'N's (ambiguous base calls) and hence contain only A, C, G and T. As we consider a k-mer and its reverse complement to be two representations of the same object, whenever we see a k-mer, we also compute the bit representation of the reverse complement and take the numerically smaller value as the unique representative of the k-mer/reverse complement pair. Note that, we only need to compute a bit representation from the whole k-mer at the start of the read. For the subsequent k-mers, just looking at the next character and applying the appropriate bit shift and bit-wise OR operations can produce the next pair of k-mer and its reverse complement. If we encounter an 'N', we have to restart the whole process from the location after 'N'. The $(k-1)$ bases skipped due to the 'N' compensates for this restart.

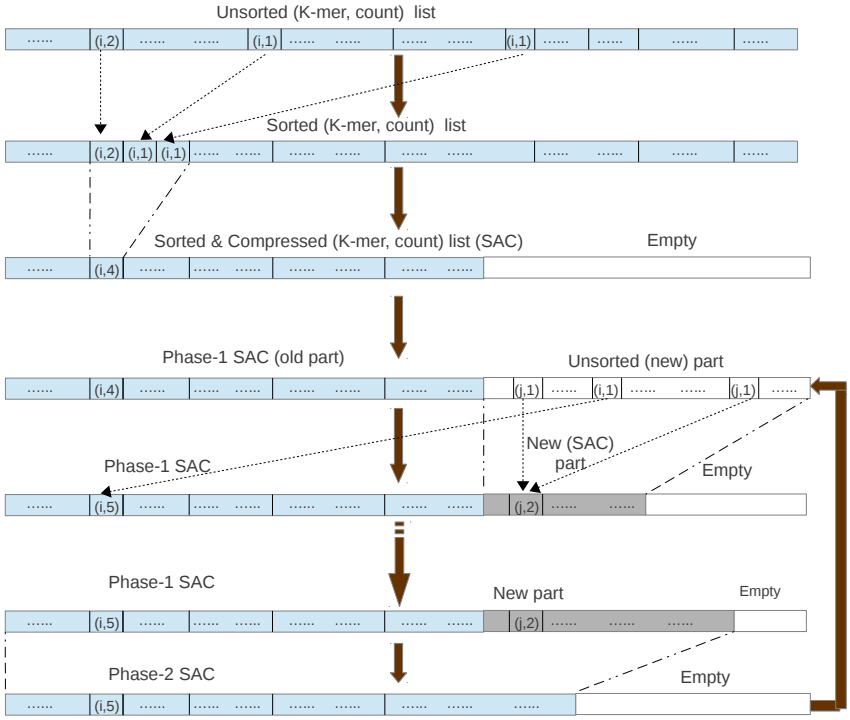
3.3.2 Identification of frequent k-mers with Pattern-blocked Bloom Filter

A Bloom filter (introduced in Chapter 2) is a space efficient probabilistic data structure which can answer set membership queries with a small false positive rate. Thus by maintaining a set of unique items seen, the Bloom filter can identify if an item was seen before in a stream of items. We use this property of the Bloom filter to identify k-mers that were seen at least twice. For improving performance, we use a modified version of Bloom filter called the Pattern-blocked Bloom filter (also introduced in Chapter 2).

3.3.3 Counting frequencies with sorting and compaction

Our next objective is to count the frequencies of the frequent k-mers. The basic idea is to store the frequent k-mers in an array A of size $> n$, where n is the number of frequent items. When this array fills up, we sort the items by the k-mer values. This places the items with the same k-mer next to each other in the array. Now, by making a linear traversal of the array, we can replace multiple items with the same k-mer with one item where a count field represents how many items were merged which is equal to how many times this k-mer was seen; see Figure 3.1. Note that, this is very similar to run length encoding. Here is a toy example: Say $A = [\ldots, (i,1), \ldots, \ldots, (i,1), \ldots\ldots, (i,1)]$. After sorting $A = [\ldots, \ldots, (i,1), (i,1), (i,1), \ldots\ldots]$ and compressing results in $A = [\ldots, (i,3), \ldots]$. We have to repeat these steps until we have seen all items. To reduce the number of times we sort the complete array, we apply the following strategy. We select a threshold $n < t < |A|$. We start with an unsorted k-mer array. It is sorted and compacted (Phase-0 Sorted and Compacted array or Phase-0 SAC). We progress in phases as follows. At phase i a certain number of items in the beginning of the array are already sorted and compressed (Phase-$(i-1)$ SAC). The new incoming k-mers are stored as unsorted items in the empty part of the array. Let m be the total number of items in the array. When $m > t$, we sort the unsorted items. Many of these k-mers are expected to exist in Phase-$(i-1)$ SAC. We make a linear traversal of the array replacing k-mers present in both Phase-$(i-1)$ SAC and the newly sorted part with one

Figure 3.1: The Sorting and compaction mechanism. We start with an unsorted k-mer array. It is sorted and compacted (Phase-0 SAC). The empty part is filled with unsorted k-mers, sorted and compacted. After repeating this step several times, the compacted new part almost fills up the whole array. Then all items are sorted and compacted to produce Phase-1 SAC. This cycle repeats until all k-mers have been seen.



item in Phase-$(i-1)$ SAC. k-mers not present in Phase-$(i-1)$ SAC are represented with one item in the newly sorted part. The counts are added up to reflect the total number of times a k-mer was seen. This takes $O(m)$ time. Note that this compaction has sequential and localized memory access which makes it cache efficient. After a few such compaction steps, $m > t$ and we sort and compress all the items in the array to produce Phase-i SAC.

By repeatedly applying this mechanism on the frequent items, we ultimately get the list of frequent k-mers with their counts decremented by 1. This is due to the fact that when inserted into the array for the first time, an item was seen at least twice unless its a false positive. To offset this, we simply add 1 to all counts before writing them out to a file.

3.3.4 Parallelization

We implemented a one producer, multiple consumer model with a pool of buffers. The producer extracts k-mers from the reads and distributes them among the consumers. Each consumer has its own Bloom filter. Since a k-mer should always pass through the same Bloom filter, we distribute the k-mers to the consumers using the modulo operation which is one of the cheapest hash functions available. Since modulo a prime number shows better hash properties compared to non-primes, it is recommended that one uses a prime (or at least an odd) number of consumers as this spreads out the k-mers more evenly among the consumers which is helpful for speeding up the parallelization. k-mers are stored in buffers and only when the buffers fill up, they are transferred to the consumer. Since consumers consume the k-mers at an uneven rate, having the same fixed buffer size for all consumers may cause the producer to block if the buffer for a busy consumer fills up. To reduce such blocking, we have a pool of buffers that has more buffers than the number of consumers. If a consumer is taking longer to consume its items, the producer has extra buffers to store its k-mers in. This improves the speed-up.

With many consumers (usually > 13), the producer becomes the bottleneck. Therefore, its important to make the producer more efficient. The two most expensive parts of the producer are: converting reads to k-mers and the modulo operation required to determine which consumer handles a particular k-mer. Modern computers support SSE (Patterson and Hennessey (1998)) instructions that operate on 128-bit registers and can parallely perform arithmetic/logic operations on multiple variables. We used SSE instructions for speeding up bit-encoding of k-mers. It is also possible to design approximate modulo functions that execute much faster than regular modulo instruction for some numbers (e.g. 5, 7, 9, 10, 63, etc) (Warren (2012)). But each of these function have to be custom designed. If we restrict the number of consumers to the numbers that have efficient modulo function, its possible to improve the producer's running time even further.

3.3.5 Running time analysis

We first analyze the sort and compress algorithm. Let the total number of frequent k-mers (those with frequency ≥ 2) be N and let n be the number of distinct frequent k-mers. We use an $xn, x > 1$, sized array A for storing the frequent k-mers and their counts. First consider the following simplified version of our algorithm: $(x-1)n$ new items are loaded into the array and they are sorted and compacted. Since there are n distinct k-mers, at least $xn - n = (x-1)n$ locations will be empty after sorting and compaction. We again load $(x-1)n$ items and perform sorting and compaction. We iterate until all N items have been seen. Each iteration takes $O(xn \log xn + xn)$ time and we have at most $N/(x-1)n$ such iterations. So, the total time required is:

$$\begin{aligned} O\left(\frac{N}{(x-1)n}(xn \log xn + xn)\right) =& O\left(\frac{x}{(x-1)}(N \log xn + N)\right) \\ \leq& O\left(\frac{x}{(x-1)}(N \log N + N)\right) \end{aligned}$$

As discussed earlier, to reduce number of times sorting is performed, which is much more expensive than compaction, we implemented a modified version of the above method which delays sorting at the expanse of more compactions. Our benchmarking shows this to be faster than the naive method. The algorithm we implemented progress in phases as follows. At the beginning of phase i, the array is filled up with unsorted elements. They are sorted and compacted ($O(xn \log xn + xn)$). This is called the Phase-$(i-1)$ SAC. Let e be the number of empty locations after each complete sorting and compaction step. Then, $(x-1)n \leq e \leq xn$. The new incoming k-mers are stored as unsorted items in the empty locations. When the empty part is full, we sort the new items ($O(xn \log xn)$). Many of these k-mers are expected to exist in Phase-$(i-1)$ SAC. We make a linear traversal of the array replacing k-mers present in both Phase-$(i-1)$ SAC and the newly sorted part with one item in Phase-$(i-1)$ SAC. k-mers not present in Phase-$(i-1)$ SAC are represented with one item in the newly sorted part. The counts are added up to reflect the total number of times a k-mer was seen. The total cost of a lazy compaction is therefore upper bounded by $O(xn \log xn + xn)$. This again creates empty locations at the end of the array which allows us to perform another round of lazy compression. We assume that the incoming items are uniformly distributed and

every lazy compaction stage reduces the size of the empty part by an approximately constant fraction $1/c$. Therefore, on average, we expect to have c lazy compaction stages. This completes Phase-i, the expected cost of which is upper bounded by:

$$\begin{aligned} & O(xn \log xn + xn + c(xn \log xn + xn)) \\ =& O((c+1)(xn \log xn + xn)) \end{aligned}$$

In order to compute how many phases are expected to consume all N items, we observe that, at every phase, the lazy compaction steps consumes a total of at least $(x-1)n\{1+(1-\frac{1}{c})+(1-\frac{2}{c})+\ldots+(1-\frac{c-1}{c})\} = (x-1)n(c+1)/2$ items. So, on average, each phase consumes at least $(c+1)n(x-1)/2$ items and hence, the expected number of phases is at most $2N/n(c+1)(x-1)$. Therefore, the total expected cost would be:

$$\begin{aligned} & \leq \frac{2N}{(c+1)n(x-1)} O\left(xn(c+1)\log xn + xn(c+1)\right) \\ & = \frac{2x}{(x-1)} O\left(N \log xn + N\right) \\ & \leq O\left(\frac{x}{(x-1)}(N \log N + N)\right) \end{aligned}$$

Note that we obtained the same expression for the naive version of sorting and compaction. It is surprising that this expression is independent of c. As an intuitive explanation, observe that more lazy compactions within a phase results in more items being consumed by a phase, which in turn, decreases the number of phases. This inverse relationship between c and the number of phases makes the running time independent of c. We found the naive version to be slower than the implemented version in empirical tests and therefore, believe our bound to be an acceptable approximation.

We now analyze the performance of sorting and compaction based strategy against a hash table based strategy for counting frequency of items. Let p be the cache miss penalty, h be the hashing cost, s is the comparison and swapping cost for sort and compress and b be the number of items that fit in the cache. The cost of frequency counting in the hash based method will be $(p+h)N$ since each hash update incurs one cache miss. For sorting and compress, we will have one cache miss for every b operations and so, the cost for sorting and compaction will be $(p/b+s)a(N \log N + N)$, where

$a = \frac{x}{(x-1)}$. To compute the value of N for which sorting and compaction will be faster than a hash based method, we write:

$$(p+h)N \geq (p/b+s)a(N\log N + N)$$
$$\log N \leq \frac{(p+h)}{(p/b+s)a} - 1$$

Let a comparison and swap be one unit of work. A conservative set of values like $s = 1, p = 160$ (Levinthal (2008)), $h = 8, b = 256$ (assuming 8 bytes items and 2KB cache), $a = 2$ results in $N \leq 2^{50}$. Therefore, for a large range of values of N, with a fast and moderate sized cache, the sorting and compaction based method would run faster than a hash based method.

Since every observed k-mer has to go through the Bloom filter, the time required in the Bloom filter is $O(M)$ where M is the total number of k-mers in the read library. So, the total running time that includes the Bloom filter checks and sorting and compression of the frequent items is $O(M) + O(N\log N + N)$. Our measurements on the datasets used show that the total time is dominated by the Bloom filter updates (i.e. $O(M) > O(N\log N + N)$).

3.4 cTurtle

For larger datasets, where $O(n)$ space is not available, the method mentioned above will fail. We show that it is possible to get a reasonable approximate solution to this problem by accepting small false positive and false negative rates. The method is based on a counting Bloom filter implementation. The error rates can be made arbitrarily small by making the Bloom filter larger. Since the count is not maintained in this method, it only reports the k-mers seen more than once (with a small false positive and false negative rate), but not their frequency.

When there are so many frequent k-mers that keeping an explicit track of the k-mers and their counts is infeasible, we can obtain an approximate set of frequent k-mers by using a counting Bloom filter. Note that, the number of bits required for a Bloom filter for n items is $O(n)$ but the constants are small. For example, it may be shown that for a 1% false positive rate, the Bloom filter size is recommended to be approximately $9.6n$

bits (Fan *et al.* (2000)). On the other hand, with a k-mer size of 32 and counter size of 1 byte, the memory required by a naive method that explicitly keeps track of the k-mers and their count is at least 9n bytes or 72n bits. With data compression techniques like prefix of k-mers being implied from the context of the data structure as in Jellyfish and KMC, this is less than 9n bytes but, from the memory comparison between Jellyfish and cTurtle presented in Table 3, we believe it still remains considerably higher than the 9.6n bits required by the Bloom filter.

The basic idea of our counting Bloom filter is to set k bits in the Bloom filter when we see an item for the first time. When seen for the second time, the item is identified as a frequent k-mer and written to disk. To record this writing, k' more bits are set in the Bloom filter. For all subsequent sightings of this item, we find the $(k + k')$ bits set and know that this is a frequent k-mer that has already been recorded. For cache efficiency, we implement the counting Bloom filter as a Pattern-blocked counting Bloom filter as follows. We take a larger Bloom filter (B) of size L. When an item x is seen, k values $(h_1, h_2, \ldots, h_k)$ within the range $[h(x), h(x) + b]$, where $h(x)$ is a hash function and b is the block size, are computed using precomputed patterns. If this is the first appearance of x, with high probability, not all of the bits $B[h_1], \ldots, B[h_k]$ are set to one and so we set all of them. When we see the same item again, we will find all of $B[h_1], \ldots, B[h_k]$ set to one. We then compute another set of locations $(h_{k+1}, h_{k+2}, \ldots, h_{k+k'})$ within the range $[h(x) + b, h(x) + 2b]$ using precomputed patterns. Again, with high probability, not all of $B[h_{k+1}], \ldots, B[h_{k+k'}]$ are set to 1 and so we set all of them. At the same time we write this k-mer to disk as a frequent k-mer. For all subsequent observations of this k-mer, we will find all of $B[h_1], \ldots, B[h_{k+k'}]$ set to 1 and will avoid writing it to disk. Note that a false positive in the second stage means that we don't write the k-mer out to file and thus have a false negative.

Currently, cTurtle reports k-mers with frequency greater than one. But this strategy can be easily adopted to report k-mers of frequency greater than $c > 1$. We argue that for most libraries with reasonable uniform coverage, $c = 1$ is sufficient. Let C be the average nucleotide coverage of a read library with read length R. Then the average k-mer coverage is $C_k = \frac{C(R-k+1)}{R}$ (Zerbino and Birney (2008a)). Suppose we have an

erroneous k-mer with one error. The probability that the same error will be reproduced when reading the same site again is $\frac{1}{3k}$ where $1/k$ is the probability of choosing the same position and $1/3$ is the probability of making the same base call error. Therefore, the expected frequency of that erroneous k-mer is $1 + \frac{C_k - 1}{3k}$. For $R = 100, k = 31$, this expression is $1 + 0.0075C$. So, we need $C > 132.85$ at a location for an erroneous 31-mer to have frequency greater than 2. Since most large libraries are sequenced at much lower depth ($< 60x$), such high coverage is unlikely except for exactly repeated regions and therefore, our choice of frequency cutoff will provide a reasonable set of reliable k-mers. However, this does not hold for Single Cell libraries which exhibit very uneven coverage (Chitsaz *et al.* (2011)), but note that, frequent k-mers are considered reliable only for uniform coverage libraries and thus single cell libraries are excluded from our consideration.

The parallelization strategy is the same as that for scTurtle.

3.5 Experiments

The datasets we use to benchmark our methods are presented in Table 4.1. The library sizes range from 3.7 Gbp to 146.5 Gbp for genomes ranging from 122 Mbp to 3.3 Gbp. Experiments were performed on a 48-core computer with AMD Opteron™ 6174 processors, clocked at 2.2 GHz, 256 GB of memory and 18 TB Raid-6 storage system and an 80-core Intel machine with Intel® Xeon® CPU E7-4870 clocked at 2.40 GHz and 1 TB of memory.

According to our experiments, with limited memory, KMC (Deorowicz *et al.*, 2013) is the fastest open-source k-mer counter. DSK (Rizk *et al.* (2013)) is also memory efficient but is quite slow. Khmer (Pell *et al.*, 2012) and BFCounter (Melsted and Pritchard (2011)) use Bloom filter based methods for reducing memory requirements. We have a similar strategy for memory reduction but achieve a much better computational efficiency.

3.5.1 Comparison with existing k-mer counters

We decided not to report times for any tool that required more than 10 hours of wall clock time and therefore some data are missing in Table 3.3. KMC was able to perform k-mer counting for all the datasets but was slower than Turtle for the larger datasets. Note that for the sake of comparison, we allowed KMC to use the same amount of memory that Turtle used but it is capable of performing the computation with smaller amount of memory. Unexpectedly, on large datasets (ZM and HS), BFCounter required more memory than scTurtle (over 128 GB vs. 109 GB). We suspect this is due to the memory overhead required to reduce collusions in the hash table used for storing frequent k-mers which we avoid using our sort and compaction algorithm. Rizk *et al.* (2013) claimed DSK to be faster than BFCounter, but on our machine, which had a 18 TB Raid-6 storage system of 2 TB SATA disks, it proved to be slower (1,591 mins vs. 1,012 mins for the GG dataset). Rizk *et al.* (2013) reported performance using more efficient storage systems (solid state disks). This might explain DSK's poor performance in our experiments. The detailed results are presented in Table 3.3 for multi-threaded Khmer, KMC, scTurtle and cTurtle. Since BFCounter (single threaded) and DSK (4-threads) do not allow variable number of threads, we present their results separately in Table 3.5.

Jellyfish's (Marcais and Kingsford, 2011) performance was inconsistent on the AMD machine (details not shown). For example, while running with 17 threads, it started with a near perfect CPU utilization of approximately 1700% but steadily declined to approximately 100% resulting in an average CPU utilization of 290% only. The computation required 16 hours and 56 minutes of wall clock time and 238 GB of memory. This is inconsistent with Jellyfish's performance on Intel machines reported in Marcais and Kingsford (2011) and Rizk *et al.* (2013). Therefore, to compare our tools to Jellyfish, we ran additional experiments on the Intel machine. Table 3.4 presents the wall-clock times for Jellyfish, KMC, scTurtle and cTurtle run with 19 threads on an Intel machine with 80 cores (Intel® Xeon® CPU E7-4870 clocked at 2.40 GHz) and 1 TB of memory for all the datasets. On this machine, Jellyfish's count step had a CPU

utilization of 1874.5% for 19 threads. We found KMC to be the fastest tool for the small datasets (DM and GG) but for the two large datasets ZM and HS, Jellyfish and cTurtles respectively were the fastest. Jellyfish had the highest memory requirements for all datasets.

3.5.2 Improving the producer

To validate our claim that the wall-clock time (and therefore parallelization) may be improved by speeding up the producer, we made special versions of scTurtle and cTurtle with are 31-threads and a fast approximate modulus-31 function. For the largest library tested (HS), on average, the special version of scTurtle (counting only) produces frequent 31-mers in approximately 73 minutes compared to approximately 90 minutes by the regular version (a 19% speedup). As we use 64-bit integers for storing k-mers of length 32 and less and 128-bit integers for storing k-mers of length in the range 33 to 64, the memory requirement for larger k-mers were also investigated. Again for the largest dataset tested (HS), we found that scTurtle's memory requirement increased from 109GB for $0 < k \leq 31$ to 172GB for $32 \leq k \leq 64$ (a 58% increase). Note that the Turtles require less memory for up to 64-mers than Jellyfish for 31-mers. Detailed results of all the datasets for the Turtles are presented in Table 3.6.

3.5.3 Error rates

We also examined the error rates for our tools and BFCounter. Note that, just like BFCounter, scTurtle has false positives only and cTurtle has both false positives and false negatives. We investigated these rates for the two small datasets (see Table 3.7) and found error rates for all tools to be smaller than 1%. For the large datasets, due to memory requirements, we could not get exact counts for all k-mers and therefore could not compute these rates.

The error rates also increase if the expected number of frequent k-mers is underestimated. As discussed in the introduction, for a genome of size g, we expect to observe approximately g frequent k-mers in the read library. To get an understanding of how

Table 3.2: Descriptive statistics about the datasets used for benchmarking. The library sizes range from 3.7Gbp to 146.5Gbp and the genome size ranges from 122Mbp to 3.3Gbp.

Set ID	**Organism**	**Genome Size (Mbp)**	**Read Lib**	**Bases (Gbp)**
DM	*D. Melanogaster*	122	SRX040485	3.7
GG	*G. Gallus*	1×10^3	SRX043656	34.7
ZM	*Z. Mays*	2.9×10^3	SRX118541	95.8
HS	*H. Sapiens*	3.3×10^3	ERX009609	135.3

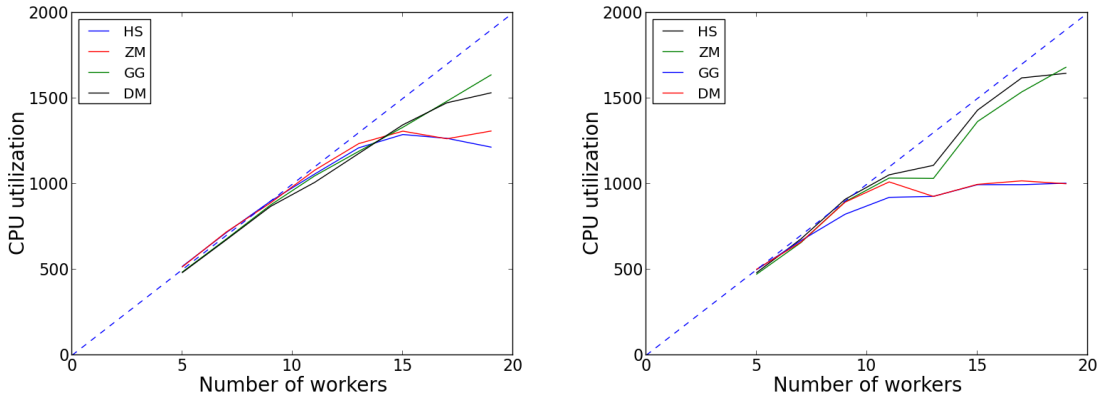
the underestimation of g drives up the error rates, we tested the DM dataset with expected number of frequent k-mers to be $g, 0.9g, 0.85g$ and found the false positive rates to be $1.87, 1.99$ and 2 % respectively. However, the true number of frequent k-mers in the DM library is $\approx 1.07g$. Therefore, we recommend setting this parameter to $\approx 1.1g$.

Software versions, commands and parameters used for producing the results presented in this paper are provided in Appendix B.

Table 3.3: Comparative results of Wall clock time and memory between Jellyfish, Khmer, KMC, scTurtle and cTurtle. Each reported number (except for the ones for for ZM and HS) is an average of 5 runs. For ZM (HS), each run of Jellyfish required approximately 3 (10) hours or more. This made multiple runs infeasible and therefore, to keep results comparable, we reported the time from one run only for all tools. The k-mer size is 31. Recall that KMC, scTurtle, Khmer and Jellyfish report k-mers and their counts, while cTurtle only reports the k-mers with count > 1.

Set ID	**Tool**	**Multi-threaded Wall clock time (min:sec)**								**Memory**
		6	8	10	12	14	16	18	20	(GB)
DM	Jellyfish	4:59	4:00	3:24	2:59	2:42	2:22	2:13	2:12	7.4
	Khmer	14:19	11:48	10:19	9:15	9:26	8:13	9:11	8:23	19.1
	KMC	7:30	5:41	3:55	3:44	3:14	3:00	2:47	2:38	5.6
	scTurtle	4:43	3:40	3:08	2:52	2:49	2:57	2:56	2:58	5.3
	cTurtle	3:41	2:43	2:04	1:55	1:55	1:55	1:55	1:56	4.2
GG	Jellyfish	60:49	48:37	41:58	37:02	34:00	32:55	30:04	28:45	81.9
	Khmer	156:03	119:22	96:59	95:37	91:41	80:13	71:20	71:17	59.7
	KMC	70:52	67:01	43:54	52:06	44:17	32:58	32:01	33:40	46.8
	scTurtle	57:52	45:09	42:48	37:07	38:59	33:15	31:28	31:29	44.9
	cTurtle	45:16	33:04	23:40	21:37	21:06	21:16	21:34	21:21	29.9
ZM	Jellyfish	295.22	375:22	361:14	357:18	349:31	370:38	326:34	296:41	158.2
	Khmer	427:14	419:16	300:13	293:00	310:02	235:56	236:19	185:00	234.4
	KMC	194:57	184:22	115:58	107:24	106:15	86:30	87:48	82:55	82.0
	scTurtle	159:49	122:42	104:22	84:48	79:25	80:39	84:26	86:31	82.1
	cTurtle	131:00	98:48	72:48	65:12	65:36	65:12	63:24	63:48	51.6
HS	Jellyfish	≥600:00	≥600:00	≥600:00	≥600:00	≥600:00	≥600:00	≥600:00	≥600:00	N/A
	Khmer	549:05	479:24	455:11	308:19	281:31	289:43	277:40	239:08	234.4
	KMC	253:18	303:8	170:42	149:22	147:40	129:47	121:14	111:27	108.8
	scTurtle	219:21	157:26	127:21	110:05	111:04	112:05	112:16	116:30	109.5
	cTurtle	171:00	123:12	98:00	87:12	91:12	89:24	88:00	90:24	68.5

Figure 3.2: CPU utilization curve for scTurtle k-mer count stage only (left) and cTurtle (right). The diagonal shows the theoretical optimum. The deviation from the optimum is largely due to the bottleneck of having a single threaded producer for extracting and distributing k-mers.



3.6 Discussion

Identifying correct k-mers out of the k-mer spectrum of a read library is an important step in many methods in bioinformatics. Usually, this distinction is made by the frequency of the k-mers. Fast tools for counting k-mer frequencies exist but, for large read libraries, they may demand a significant amount of memory which can make the problem computationally unsolvable on machines with moderate amounts of memory resource ($\leq$ 128 GB or even with 256 GB for large datasets). Simple memory efficient methods, on the other hand, can be very time consuming. Unfortunately there is no single tool that achieves a reasonable compromise between memory and time. Here we present a set of tools that make some compromises and simultaneously achieves memory and time requirements that are matching the current state of the art in both aspects.

In our first tool (scTurtle), we achieve memory efficiency by filtering k-mers of frequency one with a Bloom Filter. Our Pattern-blocked Bloom filter implementation is more time-efficient compared to a regular Bloom filter. We present a novel strategy based on sorting and compaction for storing frequent k-mers and their counts. Due to its sequential memory access pattern, our algorithm is cache efficient and achieves good running time. However, because of the Bloom filters, we incur a small false positive

Table 3.4: Comparative results of Wall clock time between Jellyfish, KMC, scTurtle and cTurtle on a SMP server with 80 cores (Intel® Xeon® CPU E7-4870 clocked at 2.40GHz) and 1TB of memory. The input of a run is a single read file (FASTA or FASTQ format) and the output is a text file containing k-mers and their frequencies (FASTA or tab delimited format). The k-mer size is 31. Each tool was run 6 times with 19 threads and the average was reported.

Set ID	**Tool**	**Wall-clock time** (min:sec)	**Memory** (GB)
DM	Jellyfish	3:01	7.4
	KMC	1:48	5.6
	scTurtle	2:12	5.3
	cTurtle	1:59	4.2
GG	Jellyfish	32:03	81.9
	KMC	17:58	46.8
	scTurtle	22:39	44.9
	cTurtle	21:31	23.9
ZM	Jellyfish	42:44	158.2
	KMC	90:36	82.0
	scTurtle	60:15	82.1
	cTurtle	58:43	51.6
HS	Jellyfish	197:42	238.0
	KMC	103:27	108.8
	scTurtle	88:31	109.5
	cTurtle	85:39	68.5

Table 3.5: Performance of BFCounter and DSK (4 threads) for 31-mers. Some of the results are not available since those computations could not be completed within a reasonable time.

Set ID	**Tool**	**Wall-clock time (min:sec)**	**CPU Utilization (%)**	**Space (GB)**
DM	BFCounter	78:35	99	3.24
	DSK	170:37	318	4.86
GG	BFCounter	1011:51	99	29.26
	DSK	1590:54	290	48.59
ZM	BFCounter	>2289:00	NA	>166.00
	DSK	>2923:00	NA	NA
HS	BFCounter	>3840:00	NA	>128.00
	DSK	>1367:00	NA	NA

Table 3.6: Performance of scTurtle (counting only) and cTurtle for 64-mers. The tools ran with fast mod and 31 threads. Each reported number is an average of 5 runs.

k-mer size	Set ID	Tool	Wall-clock time (min:sec)	CPU Utilization (%)	Space (GB)
31	DM	scTurtle	02:18	2335.8	5.50
		cTurtle	1:28	580.0	4.20
	GG	scTurtle	24:48	2388.0	47.10
		cTurtle	28:51	413.2	29.90
	ZM	scTurtle	55:57	1838.0	82.15
		cTurtle	74:46	756.0	51.60
	HS	scTurtle	73:24	1563.0	109.53
		cTurtle	98:24	512.0	68.55
48	DM	scTurtle	2:33	1790.0	8.30
		cTurtle	2:30	871.4	4.76
	GG	scTurtle	25:11	1373.8	70.69
		cTurtle	25:29	693.8	29.34
	ZM	scTurtle	90:16	1125.0	129.09
		cTurtle	81:28	782.0	52.35
	HS	scTurtle	112:11	953.4	172.11
		cTurtle	105:10	657.6	69.29
64	DM	scTurtle	1:40	948.0	8.30
		cTurtle	1:28	580.0	4.76
	GG	scTurtle	31:60	825.8	70.69
		cTurtle	28:51	413.2	29.35
	ZM	scTurtle	79:90	1037.0	129.09
		cTurtle	74:46	756.0	52.35
	HS	scTurtle	79:56	952.0	172.11
		cTurtle	98:24	512.0	69.29

Table 3.7: False Positive and False Negative rates of scTurtle and cTurtle. For the large datasets, due to memory constraints, the exact counts for all k-mers could not be obtained and therefore these rates could not be computed.

Set ID	scTurtle (FP only)(%)	BFCounter (FP only)(%)	cTurtle	
			FP (%)	FN (%)
DM	0.178	0.300	1.9×10^{-4}	2.3×10^{-4}
GG	0.848	0.027	0.31	0.08

rate.

The second tool (cTurtle) is designed to be more memory efficient at the cost of giving up the frequency values and allowing both false positives and false negatives. The implementation is based on a counting Bloom filter that keeps track of whether a k-mer was observed and whether it has been stored in external media or not. This tool does not report the frequency count of the k-mers.

Both tools allow k-mer size of up to 64. They also allow the user to decide how much memory should be consumed. Of course, there is a minimum memory requirement for each dataset and the amount of memory directly influences the running time and error rate. However, we believe, with the proper compromises, the approximate frequent k-mer extraction problem is now computationally feasible for large read libraries within reasonable wall-clock time using a moderate amount of memory.

From an algorithm development point of view, our major observation was the importance of cache-efficiency. We showed that a cache-efficient (theoretically) slower algorithm can outperform a (theoretically) faster but cache-inefficient algorithm even for large datasets. For Bioinformatic tools that frequently have to deal with large datasets, algorithms need to be optimized for performance and cache-efficiency will play an important part in achieving this. Another recent development is the adoption of external memory algorithms in Bioinformatics (e.g. the recently published k-mer counter KMC). As datasets are growing in size, algorithms can no longer afford to exclusively rely on main memory for storing their data structures and needs to adapt to take advantage of external memory.

Chapter 4

SLIQ: Simple Linear Inequalities for contig scaffolding

Scaffolding is an important subproblem in *de novo* genome assembly in which mate pair data is used to construct a linear sequence of contigs separated by gaps. Here we present SLIQ, a set of simple linear inequalities derived from the geometry of contigs on the line that can be used to predict the relative positions and orientations of contigs from individual Paired reads (PRs) and thus produce a contig digraph. The SLIQ inequalities can also filter out unreliable paired reads and can be used as a preprocessing step for any scaffolding algorithm. We tested the SLIQ inequalities on five biological data sets ranging in complexity from simple bacterial genomes to complex mammalian genomes and compared the results to the majority voting procedure used by many other scaffolding algorithms. SLIQ predicted the relative positions and orientations of the contigs with high accuracy in all cases and gave more accurate position predictions than majority voting for complex genomes, in particular the human genome. Finally, we present a simple scaffolding algorithm that produces linear scaffolds given a contig digraph. We show that our algorithm is very efficient compared to other scaffolding algorithms while maintaining high accuracy in predicting both contig positions and orientations for biological data sets.

De novo genome assembly is a classical problem in bioinformatics in which short DNA sequence reads are assembled into longer blocks of contiguous sequence (contigs) which are then arranged into linear chains of contigs separated by gaps (scaffolds). Previously, only single end short reads were available from sequencing experiments. Modern genome sequencing technology allows reporting reads in pairs commonly known as paired reads or paired end. The distance between the two reads of a pair plus the

two read lengths (the insert length) approximately follows a normal distribution determined by the library preparation protocol and may be approximately determined by mapping the pairs to sufficiently long contigs. Some genome projects also include mate pair libraries with several different insert lengths. Although there are experimental differences between mate pairs and paired-end reads, we will refer to them interchangeably as paired reads since we can treat them identically from a mathematical point of view. Paired reads are particularly important for *de novo* assembly since, in addition to building contigs, we can now hypothesize about neighbors of a contig whenever the reads of a pair fall on two different contigs. This opens the possibility of scaffolding contigs.

Computational genome assembly is typically performed in at least two stages — the contig building stage and the scaffolding stage. Here we do not address the contig building problem but rather assume that we have access to a set of contigs produced by an independent algorithm. However we discuss the relationship of the contig building and scaffolding stages later in the discussion. The scaffolding problem tries to string contigs into a chain such that the order of the contigs in the scaffold reflects their real order in the genome. For the scaffolding problem, the most popular strategy is to construct a contig graph in which nodes represent contigs and edges represent sets of paired reads connecting two contigs (i.e. the two reads of the mate pair fall in the two different contigs). The edges are given weights equal to the number of paired reads connecting the two contigs. We then try to find a walk in the graph such that the minimum number of paired reads are violated (a mate pair is violated when the contigs it is connecting do not have the relative orientation or position suggested by the mate pair). Just finding the optimal orientation assignment is reducible to the Maximum Cut problem which is known to be NP-complete (Garey, 1979). Consequently, finding the optimal walk to get the optimal scaffolding is also NP-complete. The genome can (and often does) have repeated regions; e.g. approximately 50% of human genome is accounted for by repeats (Haubold and Wiehe, 2006). But the contig builder is likely to report one contig per repeated region. This repetitive structure of the genome makes scaffolding harder as it introduces loops and cycles in the contig graph. We also have

false edges resulting from mis-assembly of reads into contigs. Unfortunately the number of false edges is not negligible and so, filtering them is a important pre-processing step.

4.1 Publication Note

The work described in this chapter was previously published in Roy *et al.* (2012). The work described is the sole work of the author, under the supervision of his PhD supervisors and in collaboration with the co-authors.

4.2 Related work

A common procedure is to filter out unreliable edges by picking a small threshold (commonly 2-5) and removing all edges with weight less than that threshold. For the remaining edges, a majority vote is used to decide on the relative orientation and position of the contigs. This simple majority voting strategy is implemented in a number of commonly-used assemblers and stand-alone scaffolders including ARACHNE (Batzoglou *et al.*, 2002), BAMBUS (Pop *et al.*, 2004), SOPRA (Dayarian *et al.*, 2010) and SOAPdenovo (Li *et al.*, 2010b) with various choices of threshold. Opera (Gao *et al.*, 2011) and the Greedy Path-Merging algorithm (Huson *et al.*, 2002) use a different strategy to bundle edges. Given a set of paired reads connecting two contigs, these algorithms calculate the median and standard deviation of the insert lengths of the set of paired reads and create a bundle using only paired reads with insert length that are close to the median. ALLPATHS (Butler *et al.*, 2008) and VELVET (Zerbino and Birney, 2008b) do not build the contig graph and thus do not have a read filtering step similar to the other assemblers mentioned. The majority voting procedure implicitly assumes that misleading paired reads are random and independently generated and that majority voting should eliminate the problematic paired reads. However, this assumption is often not true because of the complex repeat structure of large genomes, such as human.

We show that unreliable paired reads can be reliably filtered using SLIQ, a set of simple linear inequalities derived from the geometry of contigs on the line. Thus

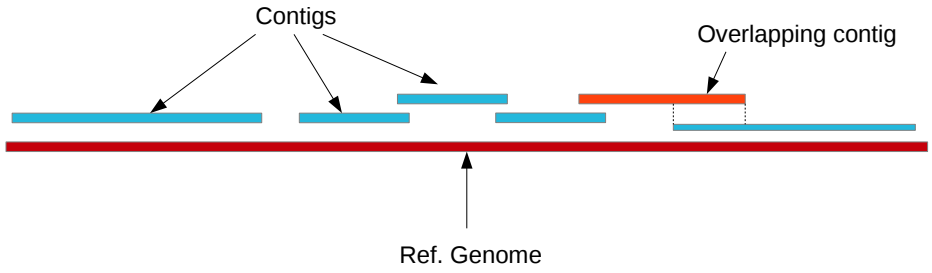
SLIQ produces a reduced subset of reliable paired reads and thus a sparser graph which results in a simpler optimization problem for the scaffolding algorithm. More importantly, SLIQ can be used to predict the relative positions and orientations of the contigs, yielding a *directed* contig graph. Our experiments show that both SLIQ and majority voting are very accurate at predicting relative orientations but SLIQ is clearly more accurate at predicting relative positions for complex genomes.

The simplicity of SLIQ makes it very easy to be integrated as a preprocessing step to any existing scaffolders, including recent scaffolders such as MIP scaffolder (Salmela *et al.*, 2011), Bambus 2 (Koren *et al.*, 2011) and SSPACE (Boetzer *et al.*, 2011). To illustrate the effectiveness of SLIQ, we implemented a naive scaffolding algorithm that produces linear scaffolds from the contig digraph. We show that despite its simplicity, our naive scaffolder provides very accurate draft scaffolds, comparable to or improving upon the more complicated sate of the art, very quickly. These scaffolds can either be output directly or used as reasonable starting points for further refinement with more complex scaffolding algorithms.

4.3 Intuition

We know that contigs can be linearly arranged on the genome. During construction, contigs get fragmented due to either lack of coverage or failure to resolve a repeated region. Neighboring contigs cannot have significant overlaps as that would dictate that they be merged into one contiguous sequence (see Figure 4.1). Each paired read that maps to two different contigs suggests a certain overlap or gap between the contigs given the insert length and the relative position and orientation. Since the gap estimation depends on the relative position and orientation of the contigs, an acceptable gap also indicates a possible relative positioning and orientation for a pair of contigs. The SLIQ equations suggests an acceptable relative position and orientation by detecting acceptable gap and if a paired read fails to provide any such configuration, it is discarded as being unreliable.

Figure 4.1: Contigs are expected to be linearly arranged on the reference genome without having significant overlap with each other.



4.4 Algorithms

We begin with a high level outline of our algorithm for constructing a directed contig graph (Algorithm 2). The crux of the algorithm is SLIQ, a set of simple linear inequalities that are used to filter paired reads and predict the relative position and orientation of contigs. In subsequent sections, we will present proofs for the SLIQ inequalities and a detailed version of the digraph construction algorithm (Algorithm 3). Finally, we will present a simple scaffolding algorithm (Algorithm 4) that uses the contig digraph to construct draft scaffolds. Throughout the remainder of this chapter, we will abbreviate paired reads as *PR*.

Algorithm 2 Construct Contig Digraph (Outline)

Require: *input:* P = a set of PRs that connect two contigs, C = a set of contigs

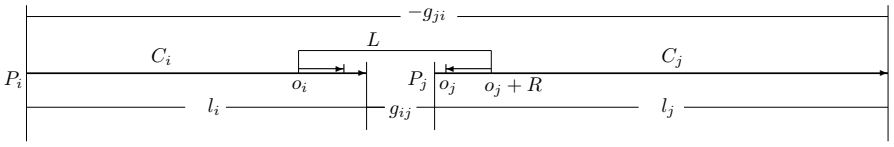
1: Construct the contig graph G with vertex set C and edges representing PRs from P that pass a certain majority cutoff.
2: Find a good orientation assignment for the contigs ($\Theta = \{\Theta_1, \Theta_2, \ldots\}$) where Θ_i is the orientation of the ith contig, for example by finding a spanning tree of G.
3: Define M_p to be the set of PRs that satisfy the SLIQ inequalities
4: Construct a directed contig graph G_d with vertex set C and edges representing PRs from M_p that pass certain criteria.

4.4.1 Definitions and Assumptions

For the sake of deriving the SLIQ inequalities, we assume that we know the position of the contigs on the reference genome. However, this information cancels out later on which allows us to analyze the PRs without access to prior contig position information.

For the derivation we also assume that all the contigs have the same orientation. Later we will not need this information.

Figure 4.2: The geometry of two contigs, C_i and C_j, arranged on a line with relevant quantities indicated. Here, L is the insert length, P_i is the start position of contig C_i, l_i is the length of the contig C_i, o_i is the offset of the read of the PR that falls on C_i, R is the Read length. $g_{ij} = P_j - P_i - l_i$. The quantities for C_j are defined similarly.



Let P_i be the position of contig C_i in the genome and l_i be the length of the contig (Fig. 4.2). We define gap g_{ij} to be the difference between the start position of contig C_j and the end position of contig C_i, and similarly for g_{ji}:

$$g_{ij} = P_j - P_i - l_i, \tag{4.1}$$
$$g_{ji} = P_i - P_j - l_j.$$

We assume that the maximum overlap of two contigs is one read length, R. In practical contig building software based on De Bruijn graphs, the maximum overlap is usually one k-mer where $R > k$ so our assumption is valid.

4.4.2 Derivation of Two Gap Equations

If we assume that $P_i < P_j$ as in Fig. 4.2 and that the maximum overlap between two contigs is R (i.e. the minimum gap g_{ij} is $-R$), then

$$P_j - P_i - l_i \geq -R,$$
$$P_j - P_i \geq l_i - R. \tag{4.2}$$

Now consider the quantity $g_{ij} - g_{ji}$. Using (4.1), we can derive the following inequality which we call Gap Equation 1

$$\begin{aligned} g_{ij} - g_{ji} &= 2\,(P_j - P_i) + (l_j - l_i) \\ &\geq 2l_i - 2R + l_j - l_i \\ &\geq l_i + l_j - 2R. \end{aligned} \tag{4.3}$$

Therefore, we have shown that $(P_i < P_j) \Rightarrow (g_{ij} - g_{ji} \geq l_i + l_j - 2R)$. Next consider the quantity $g_{ij} + g_{ji}$. We can easily derive Gap Equation 2:

$$g_{ij} + g_{ji} = -(l_j + l_i). \tag{4.4}$$

Now we will prove the other direction of the implication in Gap Equation 1 and show that $(g_{ij} - g_{ji} \geq l_i + l_j - 2R) \Rightarrow (P_i < P_j)$. Using Gap Equation 1 and Equation (4.1), we get

$$\begin{aligned} g_{ij} - g_{ji} &\geq l_i + l_j - 2R, \\ 2(P_j - P_i) + (l_j - l_i) &\geq l_i + l_j - 2R, \\ 2(P_j - P_i) &\geq 2l_i - 2R, \\ P_j - P_i &\geq l_i - R. \end{aligned} \tag{4.5}$$

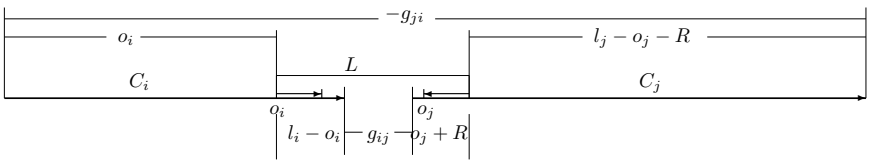
No contig length can be less than R, the length of a read. In practice, contigs of lengths R are not very reliable. Our experiments show that such contigs almost always fail to align to the reference. We suggest scaffolders enforce a minimum contig length which is $> R$. We make the assumption $l_i - R > 0$ and that gives us, $P_j - P_i > 0$ or $P_i < P_j$. Therefore, $(g_{ij} - g_{ji} \geq l_i + l_j - 2R) \Rightarrow (P_i < P_j)$ and together we have proven,

$$(g_{ij} - g_{ji} \geq l_i + l_j - 2R) \iff (P_i < P_j). \tag{4.6}$$

4.4.3 Using the Gap Equations to Predict Relative Positions

Our definitions in Equation (4.1) used the quantities P_i and P_j which are not available in practice in *de novo* assembly. Thus we need to define the gaps g_{ij} and g_{ji} in terms

Figure 4.3: The geometry of two contigs arranged on a line in terms of quantities known in *de novo* assembly.



of quantities we know such as the insert length L and the read offsets relative to the contigs o_i and o_j. Note that the insert length for each PR is an unknown constant so treating it as a constant in the proof is justified. In practice, we use $L = \bar{L} + 2\sigma$, where $\bar{L}$ is the reported or computed mean and σ is the standard deviation of the insert length distribution.

Let L be the insert length, o_i and o_j be the offsets of the start positions of the paired reads in C_i and C_j respectively and Θ_i and Θ_j be the orientations of C_i and C_j respectively. To simplify the notation we abbreviate $\Theta_i = \Theta_j$ as $\Theta_{i=j}$ and $\Theta_i \neq \Theta_j$ as $\Theta_{i\neq j}$. Then, if $P_i < P_j$ and $\Theta_{i=j}$ (see Fig. 4.3), we can redefine the gaps g_{ij} and g_{ji} without using the contig start positions P_i and P_j:

$$g_{ij} = L - l_i + o_i - o_j - R, \tag{4.7}$$

$$g_{ji} = -L - l_j + o_j + R - o_i. \tag{4.8}$$

Note that these definitions remain consistent with Gap Equation 2 (Equation (4.4)). Taking the difference of Equations (4.7) and (4.8) we can similarly remove P_i and P_j from Gap Equation 1:

$$g_{ij} - g_{ji} = 2L - 2R + 2(o_i - o_j) + (l_j - l_i). \tag{4.9}$$

Using Equations (4.9) and (4.5), we derive the following inequality:

$$2L - 2R + 2(o_i - o_j) + (l_j - l_i) \geq l_i + l_j - 2R,$$
$$2L + 2(o_i - o_j) + (l_j - l_i) \geq l_i + l_j,$$
$$L + (o_i - o_j) \geq l_i.$$

Consequently we obtain that $(P_i < P_j) \wedge \Theta_{i=j} \Rightarrow L + (o_i - o_j) \geq l_i$. Negating the implication gives

$$\neg(L + (o_i - o_j) \geq l_i) \Rightarrow \neg((P_i < P_j) \wedge \Theta_{i=j}),$$
$$L + (o_i - o_j) < l_i \Rightarrow (P_i > P_j) \vee \Theta_{i \neq j}.$$

Now without loss of generality we can assume that $\Theta_{i \neq j}$ is false. This is possible because our experiments later show that the SLIQ or majority voting procedures are both very accurate at predicting relative orientation (Table 4.3) so we can first determine the relative orientations of the contigs and flip the orientation of one contig if required. Thus we have

$$L + (o_i - o_j) < l_i \Rightarrow (P_i > P_j). \tag{4.10}$$

In addition, we introduce two filters that are very useful in practice for removing unreliable PRs. To derive the first filter, if $P_j < P_i$,

$$\begin{aligned} L &= l_j - o_j + g_{ji} + o_i + R, \\ &\geq l_j - o_j - R + o_i + R, \\ o_j - o_i &\geq l_j - L, \\ o_i - o_j &< -l_j + L. \end{aligned} \tag{4.11}$$

The second filter is to discard an PR if it passes the test for both $P_i < P_j$ and $P_j < P_i$.

4.4.4 Using the Gap Equations to Predict Relative Orientations

So far we have only predicted relative positions when $\Theta_{i=j}$. Now we show that we can also use the gap equations to infer the relative orientations of the contigs. First, if

$(P_i < P_j)$ and the minimum gap is $-R$ then we have

$$g_{ij} = L - l_i + o_i - o_j - R \geq -R. \tag{4.12}$$

Similarly, if $(P_j < P_i)$, then we define $\bar{g}_{ji}$ and write

$$\bar{g}_{ji} = L - l_j + o_j - o_i - R \geq -R. \tag{4.13}$$

Note that $\bar{g}_{ji}$ is different than g_{ji} which we defined under the assumption $P_i < P_j$ in Equation (4.8).

Since $(P_i < P_j)$ and $(P_j < P_i)$ are mutually exclusive and exhaustive neglecting $P_i = P_j$, at least one of Equations (4.12) and (4.13) will be true. Note that possibly also both could be true. For example, if $P_i < P_j$ then $g_{ij} \geq -R$. Now $(P_j < P_i)$ must be false, but that does not imply that $\bar{g}_{ji} \geq -R$ is false. If both Equations (4.12) and (4.13) are true, then we can add them to get $2L \geq l_i + l_j$. To summarize,

$$\begin{aligned} \big((g_{ij} \geq -R) \wedge (\bar{g}_{ji} \geq -R)\big) &\Rightarrow 2L \geq l_i + l_j, \\ 2L < l_i + l_j &\Rightarrow \big(\neg(g_{ij} \geq -R) \vee \neg(\bar{g}_{ji} \geq -R)\big) \end{aligned}$$

Recalling again that at least one of Equations (4.12) and (4.13) are true, we see that $2L < l_i + l_j$ is a sufficient condition for mutual exclusion (the XOR relation is denoted by $\oplus$):

$$\begin{aligned} \Theta_{i=j} \wedge (2L < l_i + l_j) &\Rightarrow (g_{ij} \geq -R) \oplus (\bar{g}_{ji} \geq -R), \\ \neg\big((g_{ij} \geq -R) \oplus (\bar{g}_{ji} \geq -R)\big) &\Rightarrow \neg\big(\Theta_{i=j} \wedge (2L < l_i + l_j)\big), \\ \neg\big((g_{ij} \geq -R) \oplus (\bar{g}_{ji} \geq -R)\big) &\Rightarrow \big(\Theta_{i\neq j} \vee (2L \geq l_i + l_j)\big). \end{aligned} \tag{4.14}$$

If we use this equation only when the PR and contigs satisfy the inequality $2L < l_i + l_j$, we can then make the relative orientation prediction

$$\neg\big((g_{ij} \geq -R) \oplus (\bar{g}_{ji} \geq -R)\big) \Rightarrow \Theta_{i\neq j}. \tag{4.15}$$

Intuitively, the condition $2L < l_i + l_j$ means that the contig lengths should be large relative to the insert length in order for the SLIQ method to work. To find contigs of

the same orientation, we arbitrarily flip one contig and run the above tests again, only this time if Equation (4.15) holds, then we conclude that the contigs were actually of the same orientation. Say we flip C_i. We call the new offset $o_{\widehat{i}}$. Then

$$\neg\big((g_{\widehat{i}j} \geq -R) \oplus (\bar{g}_{j\widehat{i}} \geq -R)\big) \Rightarrow \Theta_{\widehat{i}\neq j} \Rightarrow \Theta_{i=j}.$$

Again, we introduce two additional filters that are very useful in practical applications. First, if we find an PR that predicts both $\Theta_{i\neq j}$ and $\Theta_{i=j}$ then we leave it out of consideration. Second, if the SLIQ equations imply $\Theta_{i\neq j}$, then we require that both the reads of the PR have the same mapping directions on the contigs and similarly for $\Theta_{i=j}$.

We summarize our results in the following lemmas and Algorithm 3.

Lemma 1 *If the maximum overlap between contigs is R and $2L < l_i + l_j$, then*

$$\neg\big((g_{ij} \geq -R) \oplus (\bar{g}_{ji} \geq -R)\big) \Rightarrow \Theta_{i\neq j},$$
$$\neg\big((g_{\widehat{i}j} \geq -R) \oplus (\bar{g}_{j\widehat{i}} \geq -R)\big) \Rightarrow \Theta_{i=j}.$$

Lemma 2 *If the maximum overlap between contigs is R, the contigs have the same orientation, (i.e. $\Theta_{i=j}$), then*

$$\big(L + (o_i - o_j) < l_i\big) \Rightarrow (P_i > P_j).$$

We also summarize the SLIQ inequalities,

$$g_{ij} - g_{ji} \geq l_i - l_j - 2R,$$
$$g_{ij} + g_{ji} = -(l_j + l_i),$$
$$(g_{ij} - g_{ji} \geq l_i + l_j - 2R) \iff (P_i < P_j),$$
$$g_{ij} - g_{ji} = 2L - 2R + 2(o_i - o_j) + (l_j - l_i).$$

4.4.5 Illustrative Cases and Examples from biological data

In this section we present two illustrative cases that provide the intuition underlying the SLIQ equations. The ideal case for an PR connecting two contigs is illustrated in

Algorithm 3 Construct Contig Digraph

Require: *input:* M = a set of PRs connecting contigs, C = a set of contigs, w =cutoff weight

Define $E' = \{(C_i, C_j) : \text{an PR connects } C_i \text{ and } C_j\}$
Let $wt(i,j) =$ (number of PRs suggesting that C_i and C_j have the same orientation) $-$ (number of PRs suggesting that C_i and C_j have different orientations)
$E = \{(C_i, C_j) : (i,j) \in E' \wedge wt(i,j) \geq w\}$
Construct a contig graph G with vertex set C and edge set E.
Find a good orientation assignment ($\Theta = \{\Theta_1, \Theta_2, \ldots\}$) for the contigs, for example, by finding a spanning tree of G.
Set $M_p = \{\}$
for all $p : p \in M$ **do**
  Let C_i and C_j be the contigs connected by p.
  if $\Theta_{i=j}$ **then**
    if $\big(L + (o_i - o_j) < l_i\big)$ AND $(o_i - o_j < -l_i + L)$ **then**
      predict $P_i > P_j$
      $M_p = M_p \cup \{p\}$
    end if
    if $\big(L + (o_j - o_i) < l_j\big)$ AND $(o_j - o_i < -l_j + L)$ **then**
      predict $P_i < P_j$
      $M_p = M_p \cup \{p\}$
    end if
  end if
end for
Let $E(i,j)$ be the set of PRs from M_p that predict $P_i < P_j$ and $E(j,i)$ be the set of PRs from M_p that predict $P_j < P_i$.
Define $E_d = \{(C_i, C_j) : |E(i,j)| > |E(j,i)|\}$
Output a contig digraph G_d with vertex set C and edge set E_d.

Figure 4.4: Illustrative cases in which both reads of the PR fall in the center of the contigs (left) and the contigs have reversed positions (right).

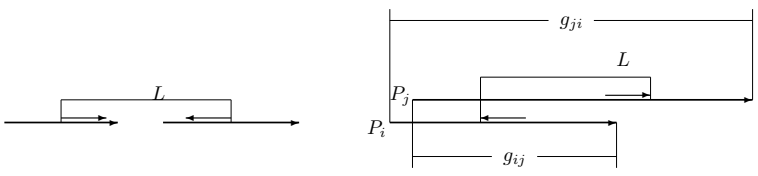


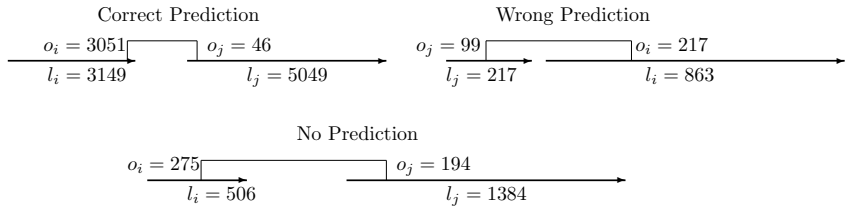
Fig 4.2. In that case the contigs are long compared to the insert length and the reads are mapped to the ends of the contigs. However, this situation does not always occur. Suppose the contigs are short such that the two reads of an PR fall exactly in the center of the contigs. Then the right hand side of Equation (4.9) reduces to $2L - 2R$. So for both cases $P_i < P_j$ and $P_j < P_i$, the right hand side of Equation (4.9) has the same value, making it impossible to predict the relative positions of the two contigs. This situation is illustrated in Fig. 4.4 on the left. It is easy to see that prediction becomes easier as the contigs get longer and the reads move away from the center of the contigs.

Now assume that the working assumption is $P_i < P_j$ but in reality, the reverse ($P_j < P_i$) is true. Then given that the contigs are long and reads map to the edges of the contigs, the insert length L would suggest the scenario depicted in Fig. 4.4 (right side). This would make both g_{ij} and g_{ji} (as calculated from Equations (4.7) and (4.8)) smaller than they should be. In reality, the position of the contigs is similar to that shown in Fig. 4.2 where we see that both g_{ij} and g_{ji} are larger than in Fig. 4.4 (right side). These wrong values would then be too small to satisfy the left hand side of Equation (4.6) and this would demonstrate that the working assumption of $P_i < P_j$ is wrong.

It is also instructive to consider examples from biological data. We show three cases from a biological data set: one in which SLIQ made a correct prediction, one in which SLIQ made a wrong prediction and one where SLIQ did not make any predictions (Fig. 4.5). We explain precisely which inequalities are violated in the figure caption. The biological examples show the difficulties of making SLIQ predictions when the reads fall close to the center of a contig or when the contig lengths are small relative to the insert size.

The ultimate aim of SLIQ is to produce better contig graphs for scaffolding and in many cases, the SLIQ inequalities successfully produced a much cleaner contig graph. In Figure 4.6, we present one such case from a *Drosophila melanogaster* dataset where a complex contig graph with loops is reduced to a chain— the desired ideal contig graph structure.

Figure 4.5: Three real examples of SLIQ predictions from the PSY dataset. For the correct prediction the equation $L+(o_i-o_j) < l_i$ evaluates to $3385 < 5043$. In the wrong prediction, it should have satisfied $L + (o_j - o_i) < l_j$ but one of the contigs is smaller than the insert length so it evaluates to $262 < 217$ (false). However $L + (o_i - o_j) < l_i$ evaluates to $498 < 863$ so the wrong prediction is made. In the no prediction case, the condition $o_i - o_j < -l_j + L$ is violated. Even if that did not fail, since one of the offsets falls almost in the center of a contig, both the conditions $L+(o_j-o_i) < l_j, (299 < 1384)$ and $L + (o_i - o_j) < l_i, (461 < 506)$ are satisfied and we would not give a prediction for this PR. To simplify the calculations we used $L = 380$.



4.4.6 Naive Scaffolding Algorithm

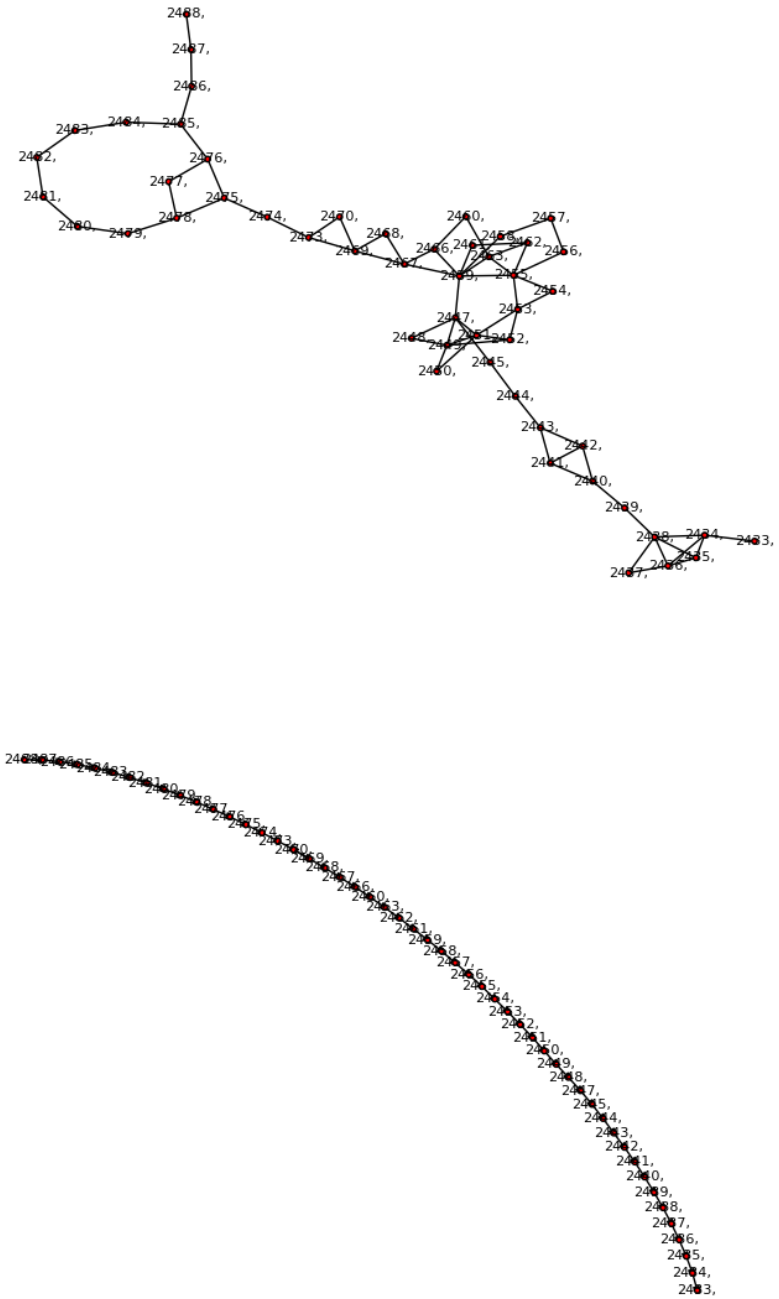
The contig digraph constructed in Algorithm 3 can be directly processed to build linear scaffolds. To illustrate this point, here we present a naive scaffolding algorithm (Algorithm 4).

Algorithm 4 Naive Scaffolder

1: $G(V, E)$ =Construct Contig Digraph (Algorithm 3)
2: Identify and remove junctions from G. Junctions are defined as articulation nodes with degree ≥ 3 that connect at least 3 subgraphs of G of size larger than some given threshold. The size of a subgraph is defined as the sum of all contig sizes in that subgraph.
3: Identify all simple cycles in G and remove the edge with the lowest weight from each simple cycle.
4: If G still contains strongly connected components, those components are removed. G is now a directed acyclic graph.
5: Output each weakly connected component of G as a separate scaffold.
6: The order of contigs in each scaffold is computed by taking the topological ordering of the nodes of their respective weakly connected component in G.

To analyze the computational complexity of the naive scaffolding algorithm, let N be the number of PRs in the library. Constructing G takes $O(N)$ time. Finding articulation points takes $O(n+m)$ time where $n = |V|$ and $m = |E|$ (Hopcroft and Tarjan, 1973). If

Figure 4.6: A component of the contig graph from a *Drosophila melanogaster* dataset is presented at the top with a minimum weight cutoff of 2 imposed on the edges. Below, we present the graph obtained by filtering the paired reads connecting the same set of contigs as on the graph on the top with SLIQ inequalities and a minimum weight cutoff of 2.



we have a articulation nodes, then finding junctions takes $O(an)$ time. Identifying and breaking simple cycles takes $O((n+m)(c+1))$ time where c is the number of simple cycles (Johnson, 1975). Finally, topological sorting takes $O(m \log m)$ time. In total the complexity of the naive scaffolding algorithm is $O(N) + O(m \log m) + O(an) + O((n+m)(c+1))$. In practical data sets, a and c are small constants and $N >> n, m, m \log m$. Thus for practical purposes the time complexity of the algorithm is $O(N)$.

4.5 Experiments

4.5.1 Datasets

To demonstrate the performance of our algorithms in practice, we ran them on five biological data sets and two synthetic data sets. The data sets represent genomes ranging in size from small bacterial genomes (3Mb) to large animal genomes (3.3Gb) (see Table 4.1 for details). More importantly, they also vary in repetitiveness– from almost non-repetitive bacteria to moderately repetitive drosophila to highly repetitive human genomes.

For each data set, we obtained a publicly available mate pair library. We used publicly available pre-built contigs for the *Drosophila simulans* (DS) and human (HS) (Gnerre *et al.*, 2011) data sets. Pre-built contigs were not available for the three microbial data sets — *P. suwonensis* (PSU), *P. syringae* (PSY) and *P. stipitis* (PST) — so we used the short read assembler VELVET (Zerbino and Birney, 2008b) to construct contigs. All software parameters and sources for the data are provided in Table 4.2. For the two synthetic datasets, *C. elegans* (SY_CE) and human (SY_HS), we constructed contigs by mapping reads back to the reference genome and declaring high coverage regions to be contigs. So, for these experiments, we have synthetic contigs but real reads. We will discuss the performance of the algorithms on the synthetic data sets at greater length in the Discussion. We mapped the reads to the contigs using the program Bowtie (v. 0.12.7) (Langmead *et al.*, 2009). Below we only report results for the uniquely mapped reads because we know the ground truth for them.

Table 4.1: Descriptive statistics about the datasets. R is the read length, cov is the coverage, L is the reported insert length, L_r is the real insert length calculated by mapping reads to the reference genome and σ is the standard deviation of L_r.

Set ID	Organism	Size.	Ref. Genome	Read Lib	R	cov	L	L_r	σ
PSU	*P. suwonensis*	3.42 Mb	CP002446.1	SRR097515	76	870x	300	188.78	18.77
PSY	*P. syringae*	6.10 Mb	NC_007005.1	(Farrer *et al.*, 2009)	36	40x	350	384.11	67.13
SY-CE	*C. elegans*	100.26 Mb	NC_003279-85	SRR006878	35	38x	200	232.13	54.44
PST	*P.stipitis*	15.40 Mb	(Chapman *et al.*, 2011)	(Chapman *et al.*, 2011)	75	25x	3.2K	3.27K	241.50
DS	*D.simulans*	109.69 Mb	NT_167066.1-68.1, NT_167061.1, NC_011088.1-89.1, NC_005781.1	SRR121548, SRR121549	36	62x	N/A	187.99	61.47
SY-HS	*H.Sapiens*	3.30 Gb	NCBI36/ hg18	ERA015743	100	45x	300	310.63	20.74
HS	*H.Sapiens*	3.30 Gb	NCBI36/ hg19	ERA015743	100	45x	300	310.63	20.74

Table 4.2: Parameter values used in the analysis of all datasets. v is the number of mismatches allowed in read mapping (Bowtie v.0.12.7).

Data Set	v	contig construction	contig mapping
PSU	2	(velvet) Hash length=21, cov_cutoff=5, min_contig_lgth=150	(vmatch) min match length $l = 150$, Hamming distance $h = 0$
PSY	0	(velvet) Hash length=21, cov_cutoff=5, min_contig_lgth=150	(vmatch) min match length $l = 150$, Hamming distance $h = 0$
PST	0	(velvet) Hash length=35, cov_cutoff= auto, min_contig_lgth=100	(vmatch) min match length $l = 200$, Hamming distance $h = 5$
SY-CE	1	(synthetic) cov cutoff=5, min contig len=L	available from synthetic construction
DS	2	AASR01000001-AASR01050477	(vmatch) min match length $l = 200$, Hamming distance $h = 5$
SY-HS	2	(synthetic) cov cutoff=3, min contig len=$2R$	available from synthetic construction
HS	3	AEKP01000001:AEKP01231194	(vmatch) min match length $l = 300$, Hamming distance $h = 0$

4.5.2 Comparison of SLIQ and Majority Voting Predictions

On all the biological data sets, SLIQ was highly accurate in predicting both relative orientation ($> 75\%$) and position ($> 80\%$) (Table 4.3). For orientation prediction, SLIQ and majority filtering produced almost identical accuracies except for the case of *P. stipitis* (PST) where SLIQ had lower accuracy (75% vs 97%). One possible reason might be that the PST library used long mate pair reads which may be more inaccurate than the other libraries we tested. Conversely, for PST, majority voting gave far worse accuracy (16.5%) than SLIQ (75%) in relative position prediction, confirming that this data set is an outlier.

Focusing only on the position predictions, SLIQ showed a significant advantage in both the number and accuracy of the predictions compared to majority voting for the more complex genomes — *D. simulans* and human (Fig. 4.7). Importantly, the improvement was particularly large for the human genome.

Finally, Table 4.4 gives a more detailed comparison of cases where the SLIQ and majority voting predictions disagreed. When the two methods disagreed, SLIQ clearly outperformed majority voting procedure. For example, for human, when the methods disagreed, SLIQ was right in 1852 cases and majority voting in only 165 cases. SLIQ was also generally more accurate when considering only the predictions made uniquely by each method, except in one case (PSY).

Table 4.3: Summary of the results of SLIQ vs. majority filtering for contig graph edges of five biological datasets. Here, n is the total number of edges connecting two different contigs, w_e is the minimum wieght of an edge for SLIQ prediction, n_o is the number of edges for which we can predict relative orientation, e_o is the accuracy of relative orientation prediction, n_p is the number of edges for which we can predict relative position, e_p is the accuracy of relative position prediction and w_m is the minimum weight of an edge for majority prediction. The same notations is used for majority filtering except with prime.

Set ID	n	w_e	n_o	e_o	n_p	e_p	w_m	n'_o	e'_o	n'_p	e'_p
PSU	4454	2	2507	99.69%	3803	99.21%	4	3942	99.59%	3925	94.87%
PSY	2086	2	1628	98.40%	1852	95.62%	4	2019	98.56%	1990	98.59%
PST	2291	1	1233	75.18%	1516	87.33%	2	1365	97.87%	1336	16.54%
DS	8738	1	6305	92.18%	7097	80.55%	2	6390	91.87%	5861	77.25%
HS	36346	1	31799	79.56%	31153	89.71%	2	32676	79.14%	25750	75.62%

Table 4.4: Comparison of position predictions between the SLIQ and majority voting methods. Here, n_a is the number of predictions where the methods agreed, n_d is the number of predictions where the methods disagreed, n_{d_e} is the number of predictions not in agreement where SLIQ was correct, n_{d_m} is the number of predictions not in agreement where majority voting was correct, n'_e is the number of predictions made only by SLIQ, e_q is the accuracy of predictions made only by SLIQ, n'_m is the number of predictions made only by majority voting, e_m is the accuracy of predictions made only by majority voting.

Set ID	n_a	n_d	n_{d_e}	n_{d_m}	n'_e	e_q	n'_m	e_m
PSU	3089	646	643	3	68	95.58%	190	90.52%
PSY	1519	287	235	52	46	86.95%	184	96.19%
PST	290	794	784	10	432	58.56%	252	25.00%
DS	2447	820	804	16	409	93.15%	2035	76.41%
HS	16425	2017	1852	165	12711	85.67%	7308	52.73%

4.6 Discussion

4.6.1 Computing the Optimal Insert Length

In our experiments, we found that using a slightly larger value for L (e.g. 20bp for PSY) than that reported or estimated increased both n_p(by 49), the number of PRs for which we could make a relative position prediction, and e_p (by 2%), the accuracy of relative position prediction. This may seem surprising at first given Equation (4.10). However, for n_p it can be seen from Fig. 4.2 that underestimating L would reduce g_{ij} which would lead to more overlaps between contigs. Since we assume that the maximum contig overlap is R, underestimating L would remove many PRs from the predictions. However, at the moment we do not have an explanation for the observed increase in e_p, the prediction accuracy.

On the other hand, using a slightly smaller value for L increased n_o, the number of PRs for which we could make a relative orientation prediction, while e_o, the prediction accuracy for orientation, remained constant. We suspect that a lower L makes Equation (4.12) and (4.13) harder to pass and thus less PRs are excluded by the mutual exclusion test.

Figure 4.7: Comparison of the accuracy of SLIQ and majority voting for relative position prediction using that same data shown in Table 4.3

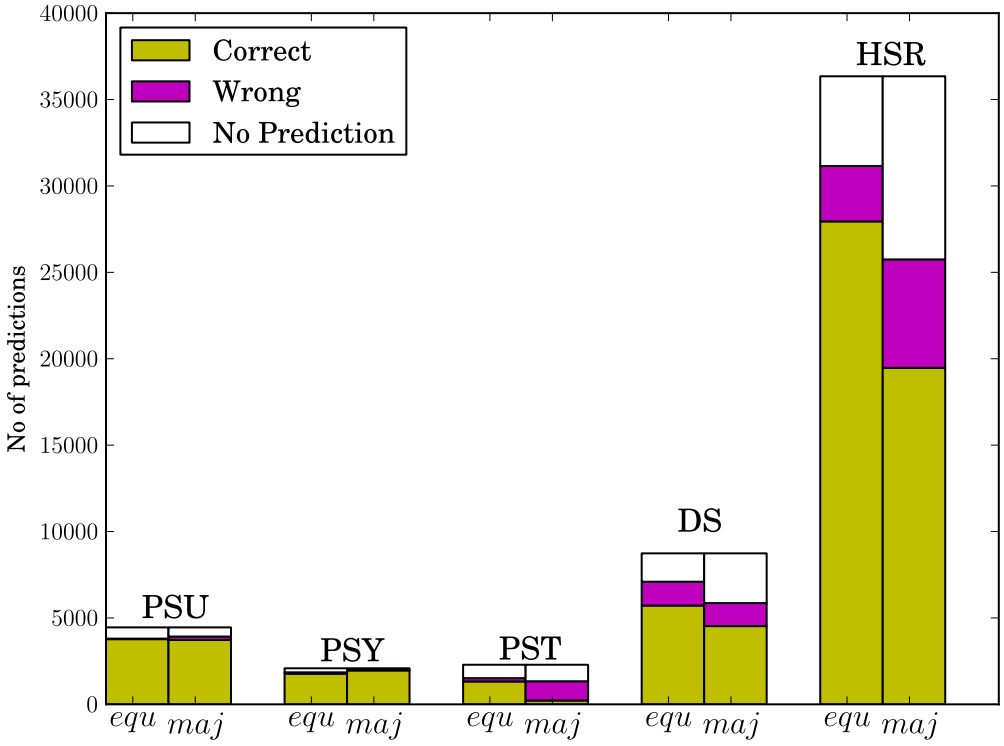
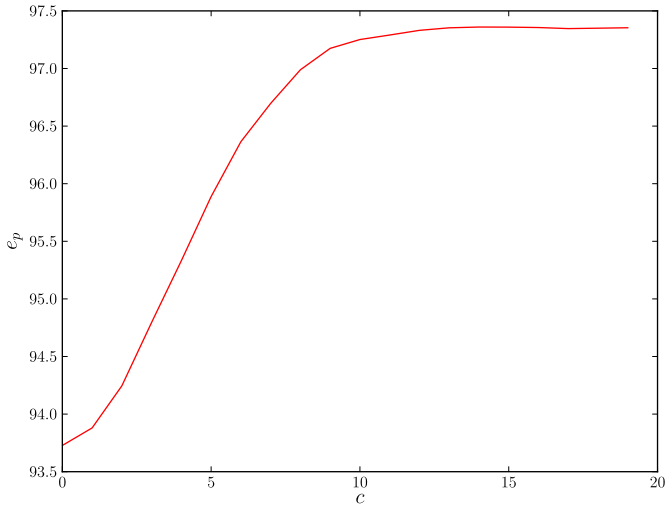


Figure 4.8: Change in the prediction accuracy, e_p, as we restrict our analysis to PRs of higher rank (c)



4.6.2 Computing the Rank of PRs

Our experimental results also agree with our illustrative cases (section 4.4.5) in that the prediction accuracy decreases as $2(o_i - o_j)$ gets closer to $(l_i - l_j)$ which intuitively means that the reads are falling closer to the center of the contigs. To address this issue we can rank the PRs by the minimum value of c for which they fail to pass the more stringent inequality $|2(o_i - o_j) - (l_i - l_j)| > cR$. We say that an PR has rank c if and only if c is the smallest positive integer such that $|2(o_i - o_j) - (l_i - l_j)| \leq cR$ and PRs with higher rank are considered more confident with regards to their prediction. Fig. 4.8 shows how the prediction accuracy depends on the rank of the PRs in the PSY dataset.

4.6.3 Effect of the Number of paired reads

More paired reads connecting different contigs give better confidence in scaffolding. But we observed that this improvement is significant up to a certain threshold (4-5

Table 4.5: Summary of the results of majority prediction for synthetic datasets for *C. elegans* (SY_CE) and humans (SY_HS). n is the total number of edges connecting two different contigs, w_m is the minimum weight of an edge for majority prediction, n_o is the number of edges for which we can predict relative orientation, e_o is the accuracy in relative orientation prediction, n_p is the number of edges for which we can predict relative position and e_p is the accuracy in relative position prediction

Data Set	n	w_m	n_o	e_o	n_p	e_p
SY-CE	17620	3	17620	99.52%	17532	99.85%
SY-HS	878380	3	878380	98.93%	868877	99.47%

Table 4.6: Summary of the results of our naive scaffolder on biological data. N50 is the length n such that 50% of bases are in a scaffold of length at least n. The position accuracy measures how many neighboring contigs in the scaffold were placed in the correct order.

Data Set	N50	Genome Coverage	Orientation Accuracy	Position Accuracy
PSU	17K	116.1%	99.64%	97.95%
PSY	75K	90.98%	98.26%	93.42%
PST	215K	97.89%	98.90%	89.89%
DS	942	59.48%	97.52%	96.07%
HS	18k	79.27%	98.28%	98.03%

for majority voting and 2-3 for the SLIQ equations). After that the improvement in correctness in scaffolding is not worth the reduction in number of edges in the contig graph. For example, for the DS dataset, if we increase the cutoff by 1, the position prediction improves by 3% but reduces the number of edges by 1520. This reduction also depends on the coverage of the read library. For the high coverage PSU dataset, an increase of 1 in cutoff has almost no effect– a reduction of 50 edges. And of course, all this is assuming that the contigs are of reasonable quality. If you have mis-assembled or chimeric contigs, more paired reads can create more loops and high degree nodes in the contig graph which are not removed by the cutoff threshold and result in worse scaffolding.

4.6.4 Performance of the Naive Scaffolder

We summarize the results of our naive scaffolder on the five biological data sets in Table 4.6 and Table 4.7. For all data sets, the orientation accuracy was very high ($> 97\%$) and the position accuracy was also high ($> 89\%$). While the genome coverages of PSU and DS may appear surprising, note that the PSU library had a very high coverage while the DS library had low coverage and was also made up of a number of different *D. simulans* strains. It is likely that the PSU contigs include misassembled fragments in the contigs, making the total length of the contigs larger than the genome size. For DS, the combination of low coverage and relatively high rates of sequence differences between the different *D. simulans* strains likely resulted in lower genome coverage.

4.6.5 SLIQ on synthetic contigs

One interesting and unexpected finding of our experiments was that the simple majority voting procedure performs very well for predicting the relative positions of contigs if the contigs have few errors. This can be seen by the performance of the majority voting procedure when using synthetic contigs that are not constructed using *de novo* assembly tools but rather by mapping the reads back to a reference genome and identifying regions of high coverage which is expected to produce much higher quality contigs (Table 4.5).

Table 4.7: Run time comparison of our Naive Scaffolder with two other state-of-the-art scaffolders, SOPRA and MIP Scaffolder. All times are the sum of the user and system times reported by the Linux `time` command. We ran all software on a 48 core Linux server with 256GB of memory.

Data Set	Naive Scaffolder	SOPRA	MIP Scaffolder
PSU	6m40.39s	237m27.237s	23m32.55s
PSY	59.36s	44m57.604s	3m14.03s
PST	67.21s	3009m29.224s	124m42.68s
DS	7m7.449s	N/A	36m42.05s
HS	241m33.928s	N/A	N/A

This observation suggests a novel way to approach the scaffolding problem in which the contig builder would output smaller but higher quality contigs and allow the scaffolder to handle the remainder of the assembly. We believe this is a significant change in philosophy of genome assembly programs to date in which during the contig building step, one generally attempts greedily to build contigs that are as long as possible. This view point also differs considerably from previous approaches to scaffolding in which the focus was on resolving conflicts between paired reads that gave conflicting information about the relative orientation and position of contigs.

Chapter 5

Single cell genome analysis of an uncultured marine stramenopile

Genomic studies traditionally works with DNA material from thousands or millions of cells obtained directly or from culture (Macaulay and Voet, 2014). Recent advances in WGA has made it possible to amplify the minute amount of genetic material from a single cell (e.g. a diploid human cell contains approximately 7 pg of genetic DNA (Macaulay and Voet, 2014)) and thereby enable their genetic sequencing. This is known as SCG where we study the genetic material from one single cell. Using this technology, single cell microbial organisms which have so far evaded genetic analysis, since they could not be brought into culture in the lab, can now be studied. Although WGA has many imperfections which may considerably effect downstream analysis, it has proven to be a very effective tool for SCG which stand poised to revolutionize many fields of biology and medicine by allowing genetic study at cellular granularity. Given a clean sample, ecologists can now study the genetic material of a unicellular microbe from the environmental, physicians can explore genetic heterogeneity in both healthy and unhealthy tissue, pathogens may be isolated and identified using genetic analysis which may greatly improve diagnosis time. In this chapter and the next, I present two projects that harness the power of SCG. This chapter presents genetic analysis of an uncultured MAST-4 straminopile and the next one proposes an improved bioinformatic pipeline for rapid pathogen detection.

A broad swath of eukaryotic microbial biodiversity cannot be cultivated in the lab and is therefore inaccessible to conventional genome-wide comparative approaches. SCG offers a promising approach whereby an individual cell is captured from nature and genome data are produced from the amplified DNA. Using this genome data, we

may gain knowledge about ecologically important protists by elucidating their genome evolution, their places in the tree of life (ToL), etc. Although this approach is widely used to analyze complete or near-complete bacterial genomes, application to draft eukaryotic genome assembly (especially from WGA single cell data) is poorly developed. However, we wanted to investigate the utility of an incomplete assembly in characterizing an uncultured organism as this, if successful, would open the possibilities of studying a very diverse range of unculturable microbial eukaryotes.

Using a single cell dataset of a marine stramenopile, we demonstrated that with a contamination free dataset, current bioinformatic tools may generate a reasonable draft assembly which can be the basis of further *in silico* analysis like protein prediction, phylogeny and metabolic pathway analysis. Our success does not imply that any unculturable organism can now be successfully studied at the genetic level, however, it certainly extends the range of possibilities of such analysis.

5.1 Publication Note

The work described in this chapter was previously published in Roy *et al.* (2014a). It focuses on the bioinformatic methods which were primarily performed by the author under the supervision of his PhD supervisors.

5.2 Related work

Multigene phylogenetic analysis using cultivated microbial eukaryotes (protists) has provided an important backbone to the eukaryote ToL (Parfrey *et al.*, 2010) but has failed to address a fundamental problem: sparse taxon sampling. This issue arises because many key lineages, and in general most protist taxa cannot be successfully cultivated (Pawlowski *et al.*, 2012; Guiry, 2012). Therefore our understanding of the protist ToL is skewed by a preponderance of data from important parasites or easily cultivated free-living lineages. Another confounding issue is foreign gene acquisition either as a result of plastid endosymbiosis (i.e., endosymbiotic gene transfer; EGT (Moustafa *et al.*, 2009; Curtis *et al.*, 2012)) or HGT from non-endosymbiotic sources (Keeling and

Palmer, 2008; Chan *et al.*, 2012; Bhattacharya *et al.*, 2013) that generates a reticulate history for many nuclear genes.

A commonly used approach to address the massive scale of microbial eukaryotic biodiversity (Pawlowski *et al.*, 2012) is DNA "barcoding" (e.g., using rDNA hypervariable regions (Behnke *et al.*, 2011)) to identify uncultured lineages. These data are however often insufficient to reliably reconstruct ToL phylogenetic relationships and do not address genome evolution. SCG shows promise in this regard and can be used to interrogate lineages on a cell-by-cell basis to understand their phylogeny and biotic interactions (Stepanauskas, 2012b; Yoon *et al.*, 2011; Bhattacharya *et al.*, 2012).

We used SCG to generate the first draft genome assembly from a cell belonging to the broadly distributed group of MAST-4 uncultured marine stramenopiles. *Ab initio* gene prediction methods were deployed to identify ca. 7,000 protein-encoding genes in the MAST-4 genome, robustly position it in the ToL using multigene phylogenetics, investigate its metabolic pathways and gain insights into its complex evolutionary history of HGT.

5.3 Sample collection and preliminary analysis

A water sample collected from Narragansett, Rhode Island, USA was sorted using flow cytometry. Single heterotrophic cells $< 10\mu$m in size lacking chlorophyll autofluorescence were retained for Multiple Displacement Amplification (MDA) prior to rDNA identification and phylogenomic analysis. Analysis of 18S rDNA sequence from one of the samples showed that it was related to uncultured stramenopiles identified in the English Channel and from Saanich Inlet in Vancouver, Canada (Figure 5.3). High sequence identity of the stramenopile rDNA to taxa within the marine stramenopile group 4 (MAST-4; e.g., accessions RA010412.25, 14H3Te6O0, RA080215T.0778) identifies this cell as a member of this abundant, globally distributed member of the plankton that consumes bacteria and picophytoplankton (Lin *et al.*, 2012; Massana *et al.*, 2004).

Figure 5.1: Phylogenetic position of the Rhode Island single cell isolate used for genome sequencing. NCBI "gi" numbers are shown for each rDNA sequence. The members of the MAST-4 clade (Lin *et al.*, 2012) are shown in red text.

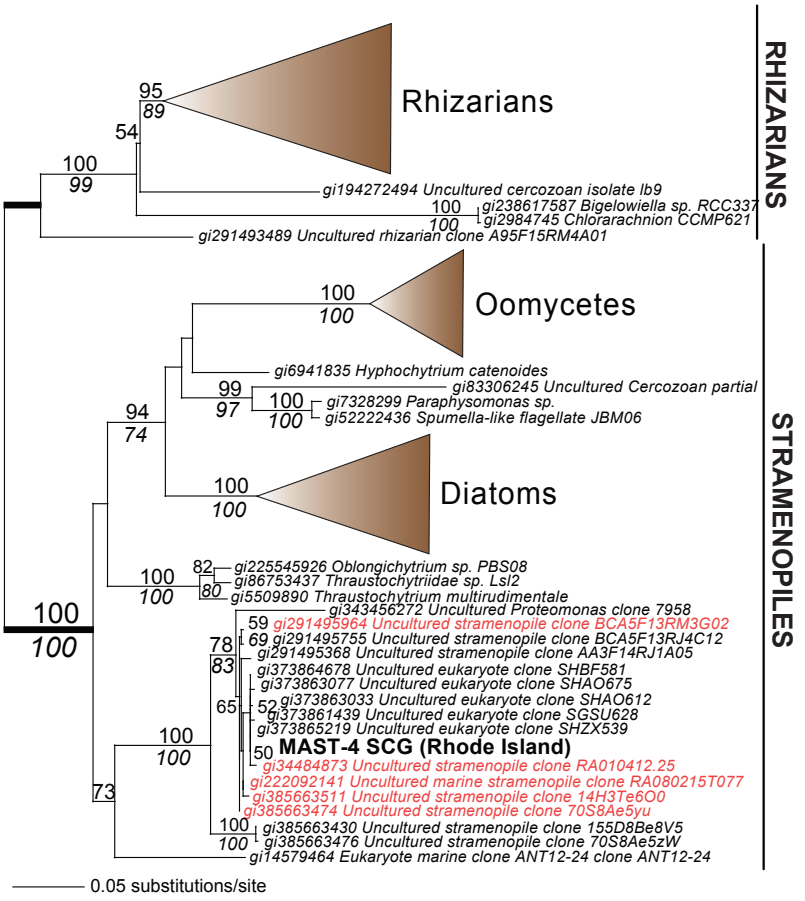
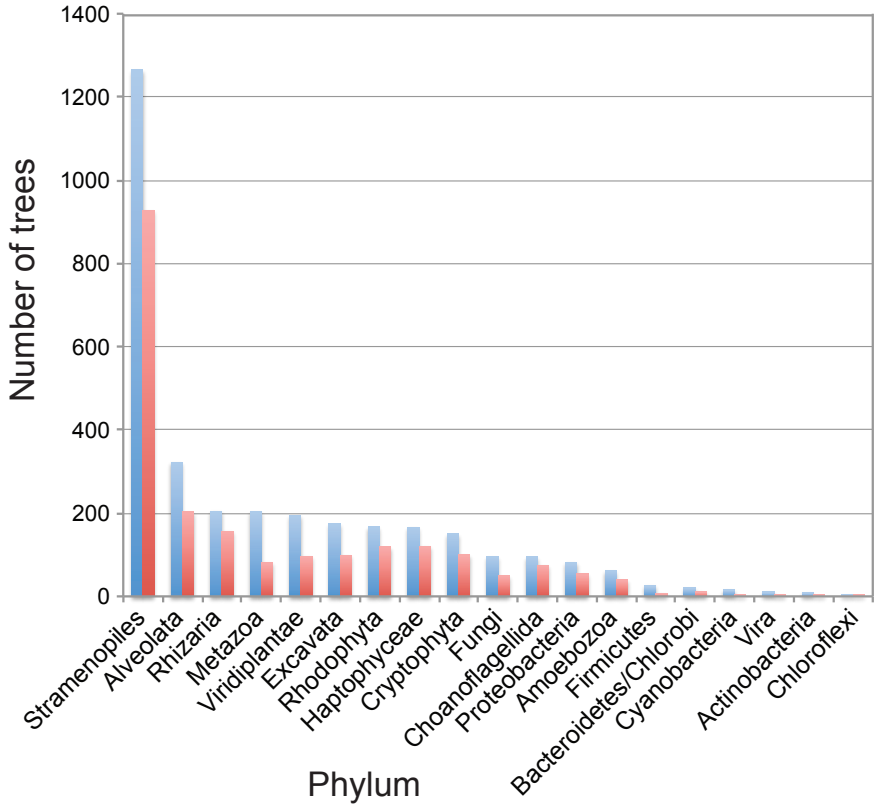


Figure 5.2: Phylogenetic distribution of all trees derived from maximum likelihood (RAxML) analysis of proteins predicted from the stramenopile MAST-4 SCG assembly are shown with the blue bars. The red bars show the number of trees in which the MAST-4 protein in the underlying alignment comprises ≥70% of the length of the reference protein at NCBI. No bootstrap cut-off was used to sort these trees.



5.4 Genome assembly and gene prediction

A total of 6.62 Gbp of Illumina paired-end reads were generated from the MAST-4 MDA sample and assembled using SPAdes 2.4 (Bankevich *et al.*, 2012). Because local alignment (using BLAST) of predicted proteins (using Augustus (Stanke and Morgenstern, 2005)) for the MAST-4 cell showed a vast majority of top hits to be eukaryotic (Figure 5.3), we expected this sample to be largely free of contaminant DNA and proceeded with further analysis.

Assembly of the MAST-4 SCG reads resulted in a 16.93 Mbp assembly. Use of the Core Eukaryotic Genes Mapping Approach (CEGMA (Parra *et al.*, 2007)) identified 159 of 458 conserved eukaryotic proteins in the MAST-4 SCG assembly. The genes encoding these proteins were then used to predict 6996 proteins using *ab initio* gene predictor Augustus (Stanke and Morgenstern, 2005), of which 3072 had a significant (e-value $\leq$1e-10) BLASTP hit with an alignment of at least 70% of the length of the protein to an existing sequence in our in-house peptide database (Supplementary Table 1). For more details on all these steps please see subsection 5.7.2.

To explore the efficacy of our assembly and protein prediction methods, we generated three independent MDA samples from total DNA prepared from a unialgal culture of the diatom *Thalassiosira pseudonana* that has a completed genome (Armbrust *et al.*, 2004) of length 32.61 Mbp. Illumina paired-end data generated from the diatom MDA samples were assembled (Table 5.1) and genes predicted (Table 5.2) as described above. The fraction of diatom reference proteins (Armbrust *et al.*, 2004) having at least 70% coverage with a BLASTP hit to the predicted proteins was 64%, 65%, 68%, and 71%, respectively in the individual and combined diatom MDA-derived genome data. In comparison, CEGMA recovered only 86.7% of the core proteins in the diatom reference assembly (that included organelle DNA) and the predicted proteins encompassed 73.40% of the reference diatom proteins with $\geq$70% coverage. Analysis of core CEGMA proteins within the predicted data from the MAST-4 SCG and diatom showed that approximately 60% and $\geq$90%, respectively, of the 458 genes were identified when mapped to the *Arabidopsis thaliana* reference. These results demonstrate that gene prediction

Table 5.1: Assembly statistics for three (A, B, C) MDA-derived diatom samples. The reference assembly is 32.61Mbp. Scaffolds with $\geq 90\%$ alignment to reference assembly were considered correct.

Dataset	**Total Data** [Gbp]	**Assembly** [Mbp]	**Correct scaffolds** [Mbp]	**No of scaffolds**	**N50** [Kbp]	**Max scaffold** [Kbp]
A	1.30	45	33.80	38245	7	101
B	0.98	41	21.43	22838	25	204
C	1.18	44	25.79	23466	17	168
Combined	3.46	48	28.48	27397	16	201

Table 5.2: Protein prediction results for the three (A, B, C) diatom MDA samples. The reference protein library has 11849 proteins. CEGMA presents 458 core eukaryotic proteins. A predicted protein is considered correct if it has a $\geq 70\%$ alignment to a reference protein. We also report the number of proteins with $\geq 60\%$ alignment to a *A. thaliana* core protein.

Dataset	**Predicted proteins**	**Correct proteins**	**Complete core proteins**	**Homologous to a *A. thaliana* core protein**
A	13523	7500	373	398
B	13022	7658	397	421
C	14933	8060	397	432
Combined	16439	8341	398	421
Reference	9413	8644	396	413

from SCG data can detect a significant number of proteins in a microbial eukaryote without an available reference genome or transcriptome data. The ca. 30% difference between the efficiency of recovery of the core set between the MAST-4 and diatom MDA samples most likely reflects the fact that the diatom DNA was derived from a culture and therefore many copies of each chromosomal region were available for WGA. In contrast, the MAST-4 SCG data were derived from a single copy of the template DNA and some genome regions were apparently not (or poorly) sampled by MDA (Yoon *et al.*, 2011; Lloyd *et al.*, 2013).

5.5 Analysis of MAST-4 and *T.pseudonana* predicted proteins using KEGG

To summarize the functions encoded by MAST-4 predicted proteins, we mapped these sequences to the Kyoto Encyclopedia of Genes and Genomes (KEGG; `http://www.genome.jp/kegg/`) categories. Given the incomplete assembly, we assumed in the KEGG analysis that missing proteins are randomly distributed across pathways. Therefore if a particular pathway is present in the MAST-4 cell, then it would be partially filled and if a pathway is absent in the KEGG analysis, then it very unlikely to be solely due to an incomplete assembly, but rather indicates true absence. Given this hypothesis, the KEGG analysis (Figure 5.3) demonstrates that many conserved pathways are present (e.g., TCA cycle), whereas others such as the urea cycle and photosynthesis are absent and presumably do not exist in the MAST-4 cell (no light harvesting chlorophyll complex proteins were found in the MAST-4 cell). Also missing are the nuclear encoded photosystem II precursors such as *psbO* and *psbM*. On the other hand, genes present in MAST-4 that are missing in the diatom include a member of glycosyltransferase family 29 involved in galactose metabolism (protein 230; EC:3.2.1.23) and several enzymes involved in amino acid and nucleotide metabolism (Figure 5.3). Analysis of proteins involved in protein synthesis [e.g., ribosomal proteins, aminoacyl-tRNA biosynthesis (Figure 5.4)] and metabolic functions [e.g., purine metabolism, fatty acid metabolism (Figure 5.5)] shows that the majority of these pathway components are present in MAST-4.

5.6 Multigene phylogenomics

Phylogenomic analysis of individual MAST-4 proteins demonstrated the expected sister-group relationship to stramenopiles (e.g., diatoms and oomycetes) and minimal evidence of contaminating DNA (i.e., 94.8% of the trees summarized in Figure 5.3 show an eukaryotic affiliation for MAST-4 proteins). To test the usefulness of these data for inferring the eukaryotic ToL, we generated a concatenated alignment from a broad collection of completed genomes that incorporated the 458 CEGMA proteins (i.e., with

Figure 5.3: Comparison of the KEGG metabolic pathway inferred from the completed genome of the photosynthetic diatom, T. pseudonana and from the MAST-4 SCG data. Pathway components present in each genome are shown with the green lines.

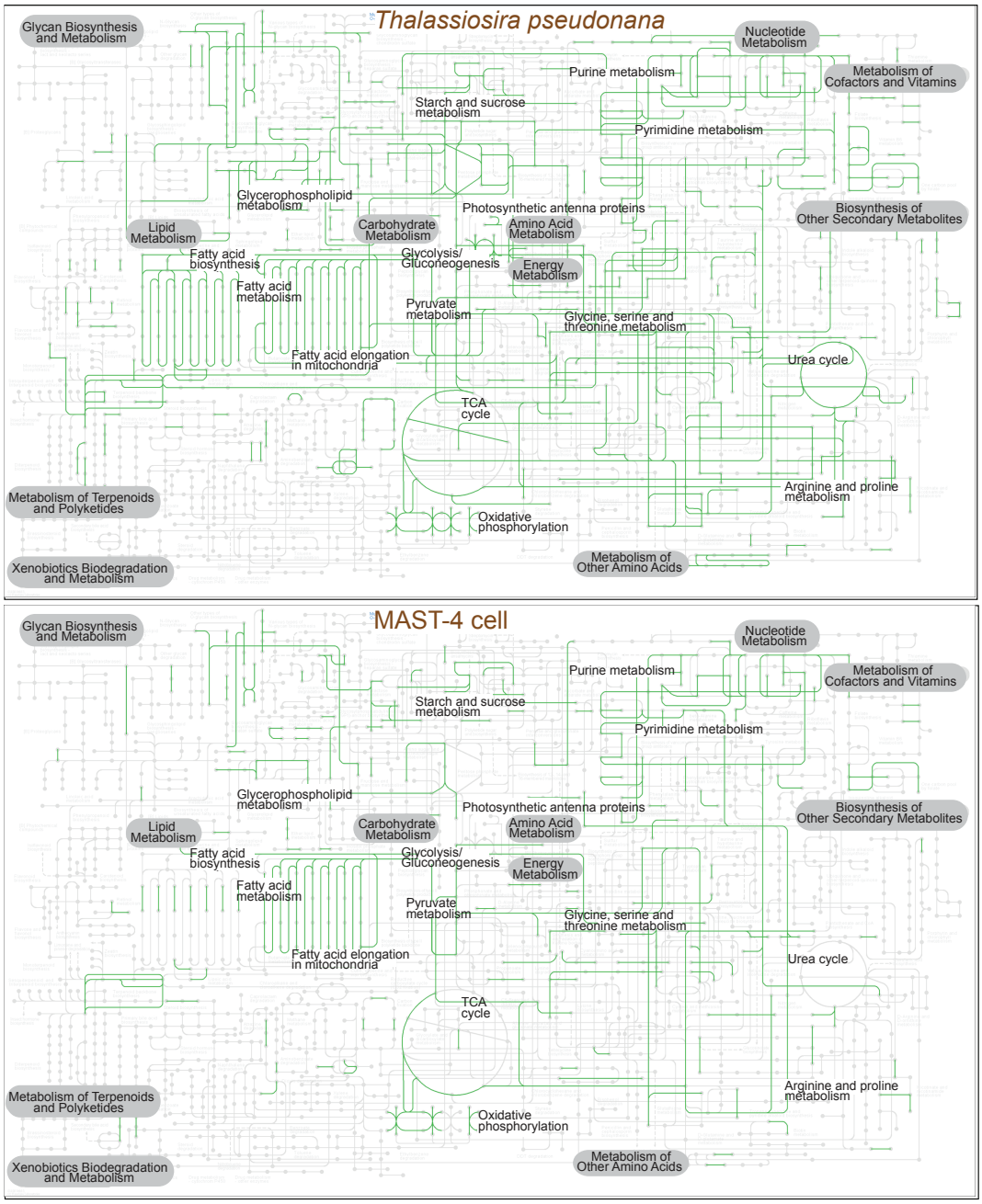
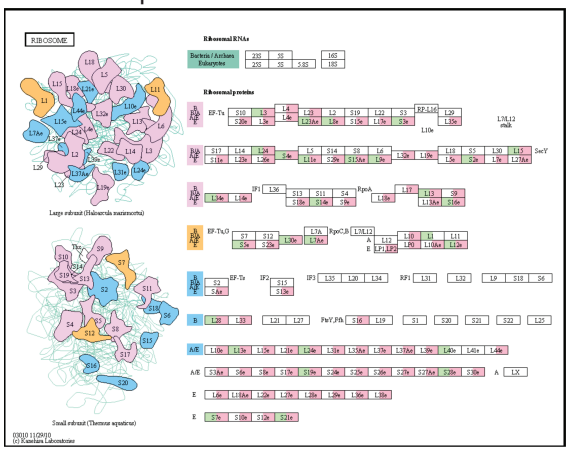
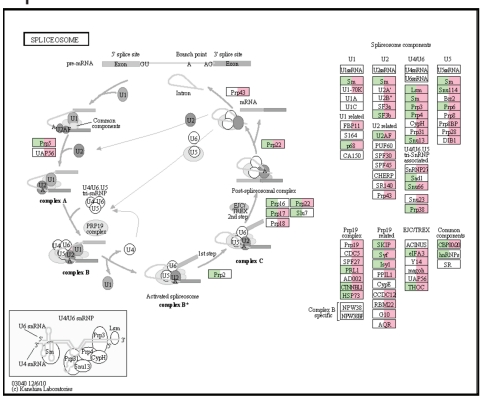


Figure 5.4: KEGG metabolic pathway map of predicted proteins from the stramenopile MAST-4 SCG that are involved in informational functions such as DNA transcription and translation (Jain *et al.*, 1999). Pathway components (proteins) present in the MAST-4 cell assembly are marked with the green boxes, whereas the red boxed mark proteins present in the reference genome assembly from the model diatom *Thalassiosira pseudonana*.

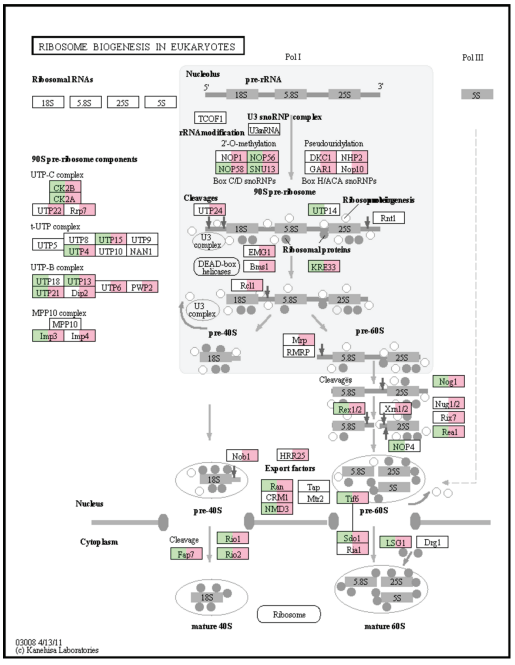
Ribosomal proteins



Spliceosome



Ribosome biogenesis



Aminoacyl-tRNA biosynthesis

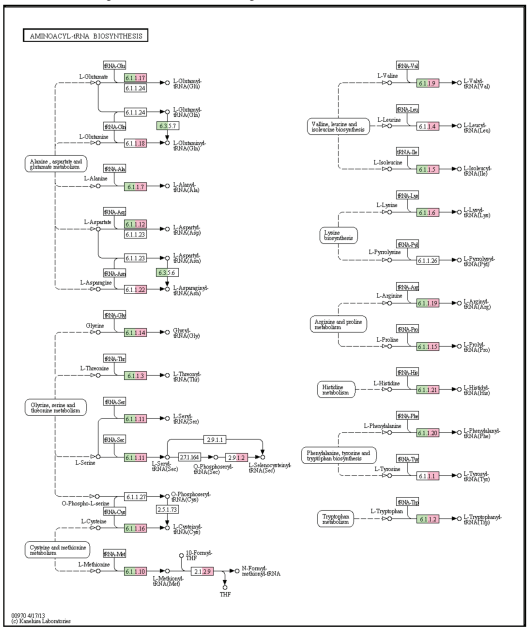


Figure 5.5: KEGG metabolic pathway map of predicted proteins from the stramenopile MAST-4 SCG that are involved in operational housekeeping functions (Jain *et al.*, 1999). Pathway components (proteins) present in the MAST-4 cell assembly are marked with the green boxes, whereas the red boxed mark proteins present in the reference genome assembly from the model diatom *Thalassiosira pseudonana*.

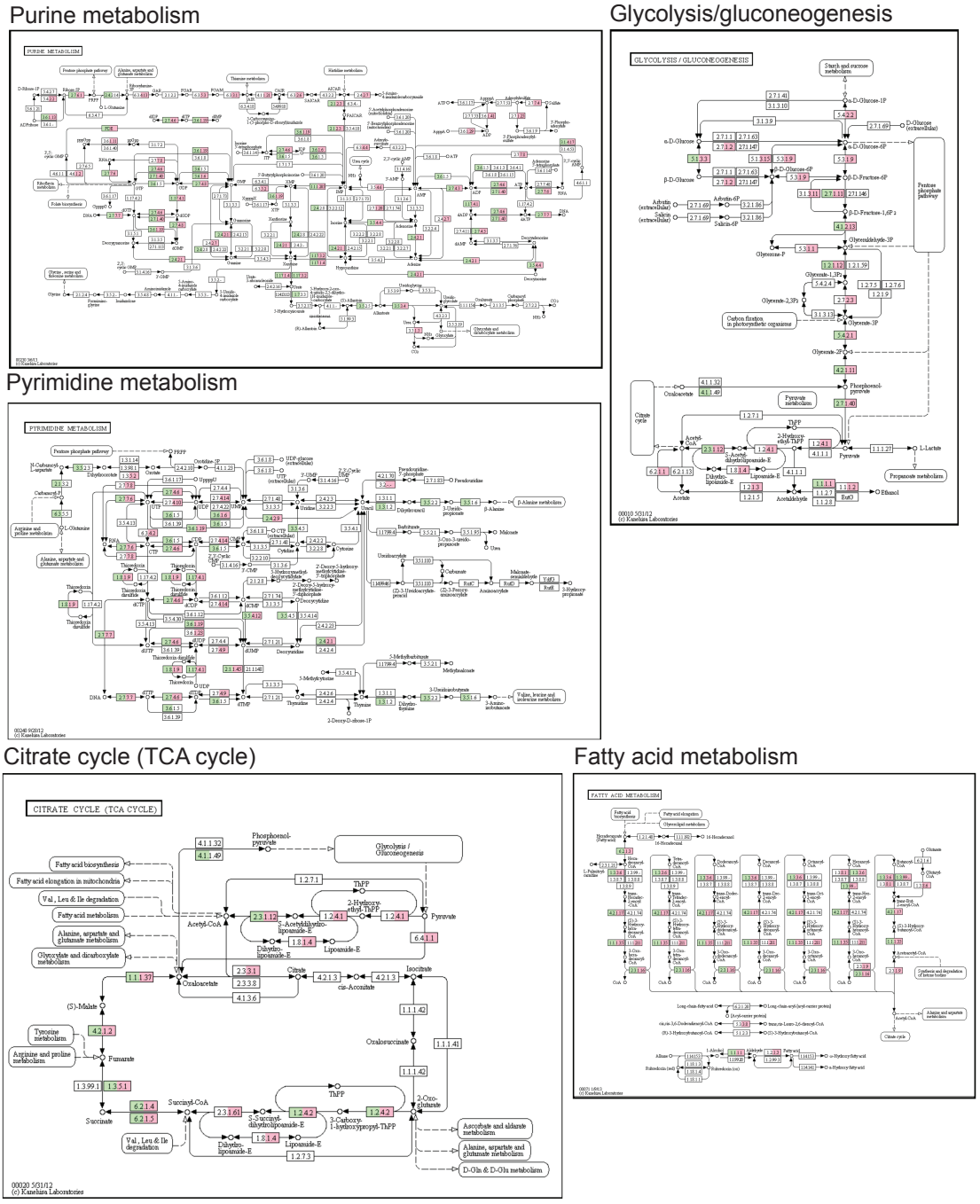
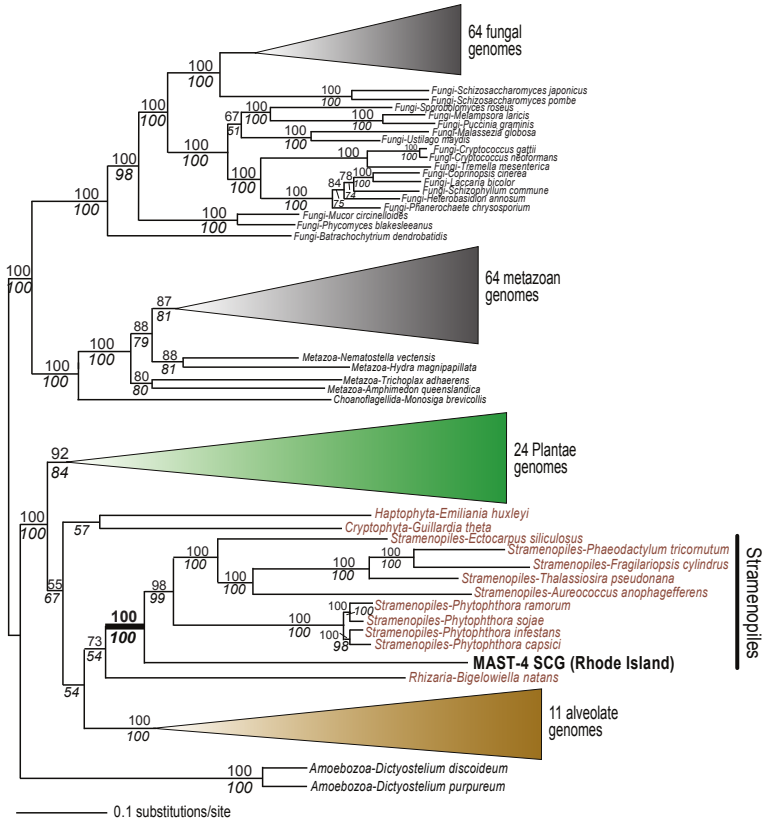


Figure 5.6: Analysis of proteins derived from MAST-4 SCG data. Phylogenetic tree inferred from the concatenated alignment of the core 458 CEGMA proteins with the results of 100 bootstrap replicates (when $\geq 50\%$) shown at the branches. The numbers in Italics below the branches derive from a RAxML bootstrap analysis using a subset of 159 CEGMA proteins that were full- length in the MAST-4 SCG assembly.



some missing data from MAST-4 and other taxa). This alignment had a length of 25,145 amino acids and maximum likelihood analysis shows 100% RAxML bootstrap support for the phylogenetic affiliation of MAST-4 with other stramenopiles in a tree that provides robust bootstrap support for many other nodes (Figure 5.6). RAxML analysis of a reduced dataset of 159 CEGMA proteins (12627 amino acids) that were complete in the MAST-4 assembly provided similar results (Figure 5.6). These results demonstrate the utility of SCG data for inferring the phylogenetic position of uncultured eukaryotes.

5.7 Methods

5.7.1 Cell collection and DNA preparation

Cell collection and DNA preparation were performed by our collaborator Hwan Su Yoon. For the sake of completeness, we are presenting the relevant section from Roy *et al.* (2014a) here.

A 500 mL sample of estuarine water was collected at low tide from the Pettaquamscutt River, Narragansett, Rhode Island, USA (41° 26' 57.32" N 71° 26' 59.49" W) on September 16th, 2009. The sample was kept in the dark and low temperature (-4° C) until processing, which was $\leq$ 6 hours thereafter. Environmental samples were prescreened through a 70 mm mesh-size strainer (Becton Dickinson). A 3 mL subsample was incubated with the Lysotracker Green DND-26 (75 nmol L-1; Invitrogen) for 10 min in order to stain protist food vacuoles using pH-sensitive green fluorescence (Rose *et al.*, 2004). Target cells were identified and sorted using a MoFloTM (Beckman-Coulter) flow cytometer equipped with a 488 nm laser for excitation. Prior to target cell sorting, the cytometer was cleaned thoroughly with bleach. A 1 NaCl solution (0.2 μm filtered and UV treated) was used as sheath fluid (Stepanauskas and Sieracki, 2007). Heterotrophic protist identification, single cell isolation, and total DNA amplifications were carried out as described by Yoon and co-authors (Yoon *et al.*, 2011). Two criteria were used for the heterotrophic cell sorting, the presence of Lysotracker fluorescence and <10 μm cell diameter. Cells were deposited into a 384-well PCR plate containing 0.6 mL of 1X TE buffer, centrifuged briefly and stored -80° C until further processing (plate no. AB108). Cells frozen in TE buffer were exposed to cold KOH for cell lysis and DNA denaturation (Raghunathan *et al.*, 2005). Cell lysate genomic DNA was amplified using multiple displacement amplification (MDA (Dean *et al.*, 2002)). The single cell amplified genomes were diluted 100-fold in sterile TE buffer, then screened by PCR and sequencing of conserved 18S rDNA. All sample 18S rDNA sequences were compared against RefSeq (`http://www.ncbi.nih.nlm.gov/refseq`) and their phylogenetic positions inferred based on a maximum likelihood tree with 100 bootstrap replications

(RAxML Version 7.2.830). One MDA sample, AB108-RIc103, was identified as a member of the uncultured MAST-4 stramenopiles and another (AB108-RIc31) was identified as a member of the cercozoan clade. Both samples were re-amplified using the REPLI-g Midi Kit (Qiagen) following the manufacturer's instructions. The second MDA reaction was treated with the QIAqick PCR purification Kit (Qiagen).

5.7.2 Single cell genome assembly

Multiple Displacement Amplification (MDA) facilitates whole genome amplification (WGA) to generate sufficient DNA from single cells for next generation sequencing (the recently described MALBAC (Zong *et al.*, 2012) procedure offers another promising approach). MDA may, however, produce significant coverage bias, resulting in fragmented and incomplete assemblies (Woyke *et al.*, 2009; Rodrigue *et al.*, 2009; Chitsaz *et al.*, 2011). Therefore, it is clear that SCG may not provide complete genome assemblies, the challenge nonetheless is to maximize the quantity and quality of the data from uncultured taxa. Given these constraints, we applied a specialized assembler to the SCG data. SPAdes 2.4 (Bankevich *et al.*, 2012) was used because it demonstrates high performance when assembling bacterial single cell libraries (`http://bioinf.spbau.ru/spades/`). Initial assembly of the read library with default settings demonstrated that despite the maximum read length of 250 bp, the median insert size was only 130 bp (with a standard deviation of 65 bp). In addition, the distribution of the library insert sizes varied widely and a significant amount of paired-end reads were of length 100-150 bp or shorter; many of the reads overlapped by 100-200 bp. In an attempt to improve the original data quality we removed reads shorter than 150 bp in length, which increased the median insert size up to 200 bp but the resulting assembly was very fragmented due to loss of coverage. The unique iterative mode in SPAdes was used to recover the usable insert length and to preserve as much coverage as possible. The program was run using k-mer lengths set of 21, 33, 55, 85, 95, and 127 in 'careful' mode. Using this approach, shorter k-mer lengths allowed us to keep the coverage and longer k-mers exploited the usable part of insert size distribution to provide proper repeat resolution. In order to predict genes we used a pipeline combining the Core Eukaryotic Gene Mapping

tool CEGMA (Parra *et al.*, 2007) and the *ab initio* gene predictor Augustus (Stanke and Morgenstern, 2005). Ab initio gene prediction tools such as Augustus need a set of training genes to "learn" gene models. Parra and co-authors (Parra *et al.*, 2007) reported a set of 458 core proteins shared by all eukaryotes. These core proteins may be used to produce a set of training examples using CEGMA. CEGMA extracted 159 (35%) core genes from the MAST-4 SCG draft assembly, which was used to train Augustus to make gene predictions. Note that, CEGMA only predicts genes for which it finds a full-length homolog. Therefore, the fragmented nature of our assembly may have been one of the major reasons for detecting only 35% of the core proteins. Interestingly the set of proteins predicted by Augustus contained 243 core proteins (with at least 60% coverage to a known *Arabidopsis thaliana* core protein), which is about 53% of the core set.

As might be expected with MDA-derived genome data, we found a considerable fraction of over-assembly of diatom reads when compared to the *T. pseudonana* reference genome. The assemblies were larger than the reference genome (32.61 Mbp) by 38%, 25%, and 35% respectively for the three independent MDA samples and 47% for the combined data. It should however be noted that SCG assemblers such as IDBA-UD (Peng *et al.*, 2012) and SPAdes 2.4 (Bankevich *et al.*, 2012) were tested and optimized for bacterial datasets and there exists no dedicated SCG assembler for larger, more complex eukaryotic genomes. Therefore, we were not expecting a perfect or near-perfect assembly from either the reference diatom or the MAST-4 genome. As such, we strove to generate as complete a set of predicted genes as possible from the fragmented assembly.

5.7.3 Phylogenomics

Phylogenomic analysis was performed as described in (Moustafa *et al.*, 2009; Bhattacharya *et al.*, 2013; Price *et al.*, 2012). Briefly, the MAST-4 predicted proteins were used in a BLASTP query against an in-house peptide database consisting of ca. 16.9 million sequences derived from RefSeq v.51 with the addition of sequenced eukaryote (e.g.,

Fungi, Metazoa, Vididiplantae, and stramenopiles) taxa from the Joint Genome Institute (`http://www.jgi.doe.gov`) and 6-frame translated eukaryote EST sequences retrieved from NCBI dbEST (`http://www.ncbi.nlm.nih.gov/dbEST`). A taxonomically diverse set of target peptides were selected, aligned via MAFFT v641, and used for phylogenetic reconstruction under the PROTGAMMALG model of RAxML v.7.2.830. The resulting trees were sorted for different patterns of monophyly with PhyloSort (Moustafa and Bhattacharya, 2008). The number of trees having the MAST-4 protein and a particular phylum in the same monophyletic group (Figure 5.2) shows a strong phylogenetic relationship between the single cell isolate and stramenopiles. From these distributions, a set of possible taxa that MAST-4 might be most closely related to was selected. The chosen phyla were: Viridiplantae, stramenopiles, Rhodophyta, Rhizaria, Opisthokonta, Metazoa, Haptophyta, Glaucophyta, Fungi, Cryptophyta, Choanoflagellida, Alveolata, and Amoebozoa. The remainder of the phylogenetic analysis was conducted with species from these phyla only. We used a method similar to Robbertse and co-authors (Robbertse *et al.*, 2011) for building a species tree. Briefly, in this method the objective is to create a super-protein alignment by concatenating all the alignments of the proteins shared among the species and to create a phylogenetic tree using that super-protein alignment. The resulting tree is considered to represent the species tree.

This procedure was performed as follows: the 458 conserved eukaryotic proteins were aligned to our local protein database to identify homologs. Note that we used 159 conserved eukaryotic proteins as references from the MAST-4 SCG assembly and for the remainder we used the conserved proteins in *A. thaliana*. For each conserved protein, a multiple sequence alignment of the conserved protein and its homologs was produced using MUSCLE (Edgar, 2004). Such alignments often contain non-conserved blocks and it has been argued that using only conserved blocks produce a more reliable phylogenetic tree (Talavera and Castresana, 2007). Therefore, Gblocks (Talavera and Castresana, 2007) was used to extract conserved regions from the alignments and these aligned conserved blocks were concatenated to produce the super-protein. This has the added benefit of making the super-protein alignment shorter, which helps in more rapid phylogenetic tree construction. The resulting super protein alignment was 25,143 amino

acids in length. Lastly, the super protein alignment was used to build a phylogenetic tree using RAxML (Stamatakis, 2006) with 100 bootstraps. This tree was considered to be the species tree for the MAST-4 SCG (Figure 5.6).

5.8 Discussion

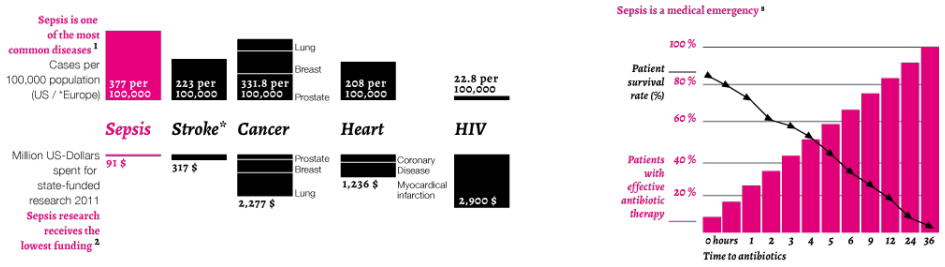
SCG is a rapidly developing field in medical and environmental science. In the latter area, virtually all work done until now has focused on prokaryotic taxa that have smaller, less complex genomes compared to eukaryotes. Prokaryotes are more amenable to genome assembly when using MDA-derived sequence data. The initial work on marine protist SCG showed complex biotic interactions for taxa (e.g., presence of prey and pathogen DNA in the MDA sample) that precluded a robust assembly of the host nuclear genome. Here we show that if a MDA sample derived from the natural environment is largely free of contaminating DNA, that it is possible to generate a useful, partial draft assembly from the cell. Applications of these SCG data include generating a gene inventory for the uncultivated taxon to study metabolic pathways and placing it in the ToL using multigene phylogenetics. Our results suggest that SCG data can be used to create a draft genome assembly for uncultured eukaryotes which, after *ab initio* gene predictions, can facilitate its positioning in a phylogenetic tree, explore its metabolic pathways and investigate its evolutionary history. With regard to inferring the ToL, we show that the CEGMA proteins appear to be well suited for training *ab initio* gene prediction tools and for phylogenetic analysis of novel organisms even in the face of significant HGT. Together, this opens up the possibility of creating a well-sampled ToL without the need for culturing organisms in the laboratory. Generation of protist (particularly picoeukaryotic) marine metagenome data should provide the opportunity to elucidate the complex history of gene sharing via HGT and the global distribution of microbial eukaryotes (Moustafa *et al.*, 2009; Curtis *et al.*, 2012; Keeling and Palmer, 2008; Bhattacharya *et al.*, 2013; Frommolt *et al.*, 2008; Archibald *et al.*, 2003; Andersson, 2009).

Chapter 6

Rapid pathogen detection with Single Cell Genomics (SCG).

The desire to harness the power of genetics to improve clinical diagnosis is one of the major driving forces behind improvements in sequencing and bioinformatics. Recent developments in SCG has opened up new possibilities in genetics and researchers are trying to apply it for improving the state of the art in various fields. One such area is the identification of disease causing agent(s) and understanding its antimicrobial susceptibility as this is very important for positive patient outcomes. In cases like sepsis, where the mortality rate rapidly increases with time, rapid identification of the disease causing pathogen is of utmost importance and may be the deciding factor in a patient's outcome. Though sepsis is one of the most common fatal diseases, unfortunately, it receives very little in terms of publicly funded research (Figure 6.1). Whole genome sequencing is becoming an increasingly important tool for pathogen detection and understanding pathogenicity and possible antibiotic resistance. Single Cell Genomics (SCG) can be a powerful alternative to the time consuming step of pathogen culturing. Moreover, for the vast majority of pathogens that cannot be cultured, SCG may be the only option for genetic analysis based pathogen detection. Currently, bioinformatic analysis of sequencing data is performed after completion of sequencing. We propose real-time bioinformatic analysis that is performed simultaneously with sequencing to facilitate rapid clinical response. To demonstrate the feasibility of our approach, we show preliminary results with an MDA amplified *E. coli* dataset.

Figure 6.1: Some facts about sepsis. While being one of the most common diseases, sepsis receives very little research funds while being one of the most deadly. Source:`http://www.wfpiccs.org/projects/sepsis-initiative/`.



6.1 Related work

In an ideal scenario, when diagnosing a disease, we should be able to identify the disease causing pathogen and also predict its drug resistance in a timely manner so that appropriate measures can be taken as soon as possible. Traditional clinical methods such as staining and microscopy, culturing can take an extended period of time while still producing ambiguous results. For example, blood cultures for sepsis, which take between 1-5 days (Ecker *et al.*, 2010), are reported to be negative in more than 50% of the cases where true bacterial or fungal infection is believed to exist (Dellinger *et al.*, 2008). For this reason, it is recommended that the treatment of sepsis be based on clinical judgment rather than culture results. Thus, blood cultures are not a satisfactory gold standard for bloodstream infections. DNA molecule based (also known as *molecular diagnosis*) methods like *SeptiFast* (from Roche) identify more positive specimens compared to blood culture methods, were often proved to be clinically relevant based on chart review and clinical data, and many of them were later confirmed by isolation of the pathogen from relevant clinical samples (Struelens, 2010). *SeptiFast* reportedly identifies the 25 organisms that account for more than 90% of the culturable pathogens associated with sepsis. Though SeptiFast is limited to culturable pathogens, its analysis time is only 6 hours and therefore is much faster than blood-culture results. Thus the sensitivity and speed of molecular diagnosis tools clearly establishes them as a preferred tool of diagnosis of bloodstream infections. SeptiTest (from MolZym) and

LOOXSTER/VYOO (from SIRS-Lab) are two more diagnosis tools of this family.

Polymerase chain reaction (PCR) amplification coupled with Electrospray ionization/Mass Sepctrometry (PCR/ESI-MS) was initially developed for identification of microbes, including previously unknown or unculturable organisms from samples that contained multiple species, primarily for biodefense (Ecker *et al.*, 2005) and is now being adapted to pathogen detection (Ecker *et al.*, 2010). This method uses primers designed to identify highly conserved genomic regions (e.g. 16s rDNA, housekeeping genes like *tufB, prlB, valS,* etc.) across bacterial or fungal domains. The mass spectrometry effectively weighs the PCR amplicons with sufficient accuracy so that the composition of A, C, G and T can be deduced. This composition is then compared to database of base composition signatures of known organisms derived from previously determined PCR/ESI-MS. The advantage here is that a probe designed from the target nucleic acid sequence is not needed— the amplicon is weighed, the base composition fingerprint is computed and compared to all known organisms in the database. Thus PCR/ESI-MS is equivalent to running thousands of specific tests, including those that have not yet been developed, since the exact identity of the organism is not required for this method. In addition to pathogen identification, drug-resistance may also be interrogated by searching for markers like *mecA* for β-lactam antibiotic resistance, *vanA* and *vanB* genes for vancomycin resistance and the bla_{KPC} gene for resistance to the carbapenem class of antibiotics. PCR/ESI-MS systems are available commercially and are being used for research and disease surveillance at 20 cites in Europe and the USA (Ecker *et al.*, 2010). Currently, the complete process from sample collection to report generation can be completed within 4-6 hours. The mass spectrometer can analyze an amplification produce in approximately 30 sec and thus can analyze approximately 3000 PCR reactions in 24 hours in a completely automated fashion. It does not even require any training in Mass spectrometry (MS) for personnel who operate the instrument or for those who interpret the results. One major limitation of PCR/ESI-MS is that only three types of drug-resistance can be investigated with current technology. Understanding resistance to other type of drugs require analysis of too many regions of the genome to be included in the current assay format.

Another important and relevant issue is responding to diseases like Tuberculosis (TB). Though well known treatments of TB exist, diagnosis often delays appropriate response. Bacteria culture is currently the gold standard for TB (*M. tuberculosis*) detection due to its highest sensitivity ($< 60\%$), but requires weeks to obtain results (Liong *et al.*, 2013). Such delayed case detection is associated with reduced cure rates and provides opportunity for continued transmission, which became an even more serious problem with the co-infections of HIV and the emergence of highly drug-resistant TB (Pawlowski *et al.*, 2012). Moreover, when diagnosing active TB infections by staining the patients sputum for acid-fast positive bacilli, we may observe false-positives in the presence of alternative mycobacterium species in patients that have immuno-modulary diseases such as AIDS/HIV or are chronic smokers (Roth *et al.*, 1997). Developing a platform for fast and improved TB detection is thus considered crucial for efficient TB control. Liong *et al.* (2013) reported a magnetic barcode based genetic detection method for TB that can detect *M. tuberculosis* and also identify drug-resistant strains in 2.5 hours from mechanically processed sputum samples. However, this method needs systematic designing of magnetic barcode probes for the pathogen under investigation which means that the identity of the pathogen and its drug-resistant strains has to be anticipated and therefore, generalizing this method for rapid pathogen detection for novel cases is non-trivial.

WGS is rapidly becoming an important, alternative tool for the diagnosis of infectious disease (van Belkum *et al.*, 2013; Liong *et al.*, 2013; Ecker *et al.*, 2010) in such complex scenarios. As our understanding of the molecular nature of microbial drug resistance improves enabling us to better predict which drug will be the most effective, whole genome/exome sequencing based methods will increasingly replace conventional culture based drug-susceptibility experiments. We can also envision using WGS to continuously monitor and analyze specimens at various stages of antimicrobial resistance and thus better understand the complex process of developing drug-resistance. Though sequencing data can provide desperately needed improvements, it is still lacking in speed of discovery because pathogens need to be cultured to obtain sufficient amounts of DNA for sequencing. Single cell genomics allows us to bypass the time consuming

culturing stage as follows. Fluorescence-activated cell sorting (FACS) or micro-fluidic methods can isolate individual cells and their genetic material can be extracted. The minute amount of DNA obtained from a single cell can be amplified using WGA (Lepere *et al.*, 2011) methods like MDA (Lasken, 2007; Worden *et al.*, 2011) or the MALBAC procedure (Zong *et al.*, 2012; Stepanauskas, 2012a) to an amount that can be used for High Throughput Sequencing (HTS). Since, WGA processes like MDA amplifies DNA material very unevenly from the various regions of the genome, the genome coverage is highly variable (coverage refers to the number of sequence reads that align on average beneath each nucleotide in an assembled piece of DNA; higher, uniform coverage is better). This high degree of non-uniformity makes separation of genomic (error free) sub-sequences from erroneous ones difficult. Therefore, at this point, the challenge for the community is to design and implement advanced bioinformatic methods and tools which can handle WGA sequencing data. We want to push even further and investigate whether the bioinformatic analysis can be performed in parallel with the sequencing experiment. Using an MDA amplified *E. coli* dataset, we demonstrate that with only 25bp partial reads, a vast majority of the proteins could be predicted which implies that this organism could have been identified with partial sequencing data extracted from an ongoing sequencing experiment.

6.2 Identifying reliable sequencing information

The fundamental problem that we are trying to address is the separation between genomic (error-free) and erroneous sequences from WGA libraries. Our analysis is based on k-mer frequencies. A k-mer is any sequencing read subsequence of length k. As discussed earlier in Chapter 3, k-mers play an important role in many methods in bioinformatics because they are at the core of the *de Bruijn* graph structure (Pevzner *et al.*, 2001) that underlies many of today's popular *de novo* assemblers (Simpson *et al.*, 2009; Zerbino and Birney, 2008a). They are also used in assemblers based on the overlap-layout-consensus paradigm like Celera (Miller *et al.*, 2008) and Arachne (Jaffe *et al.*, 2003) as seeds to find overlap between reads. Several read correction tools (Kelley *et al.*, 2010; Liu *et al.*, 2012; Medvedev *et al.*, 2011) use k-mer frequencies for error

correction. These methods were developed for multi-cell libraries where the coverage is approximately uniform and therefore, are not applicable to single-cell libraries with highly uneven coverage.

6.2.1 Multi-cell vs Single-cell libraries

For multi-cell libraries, which exhibit a near uniform genome coverage, k-mers which appear multiple times can be assumed to reflect the true sequence of the donor genome since it is very unlikely that random errors will generate the same k-mer multiple times. Thus, k-mers which only appear once are assumed to contain sequencing errors (see Figure 6.3, left). Therefore, high frequency k-mers constitue a high quality representation of the sample genome.

Unfortunately, due to the large coverage variation introduced by WGA methods, such assumptions do not hold for amplified single-cell libraries. Regions of high coverage can produce the same erroneous k-mer multiple times just by chance (see Figure 6.3, right) and such high coverage regions are very common in WGA libraries (see Figure 6.2 (Chitsaz *et al.*, 2011)), producing a large number of erroneous k-mers that appear multiple times. We refer to the erroneous frequent k-mers as *false frequent* and the error-free genomic frequent k-mers as *true frequent*. For downstream analysis such as assembly, having a large number of false frequent k-mers usually results in very fragmented contigs. Therefore, minimizing the proportion of false frequent k-mers while ensuring a (near) complete set of true frequent ones is desirable.

6.2.2 Intuition

In order to counter the high coverage bias and obtain a reasonable coverage of the genome, single cell read libraries typically have a high average coverage. Therefore, partial reads (a fixed length prefix of the reads) are also expected to have sufficient average coverage. Though using partial (shorter) reads would lead to a poorer genome coverage and worse assembly contiguity, we are interested in exploring whether this is sufficient for identifying a reasonable number of genes and understand an organism's pathogenic capabilities. Also, since the error rate usually increases towards the end

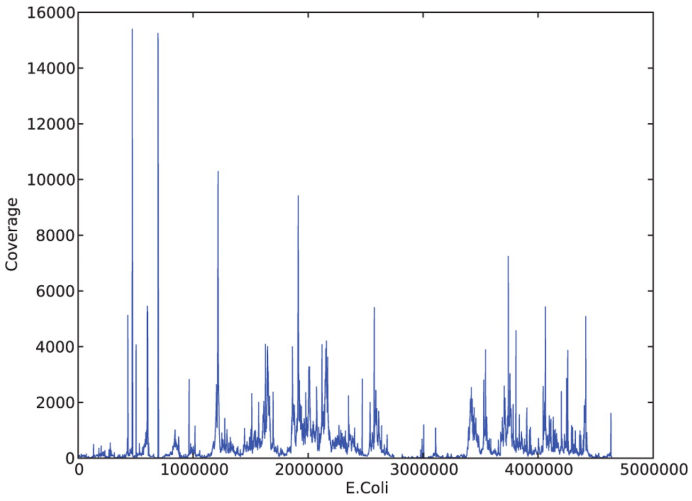


Figure 6.2: Highly uneven genome coverage observed for MDA amplified *E. coli* read library.

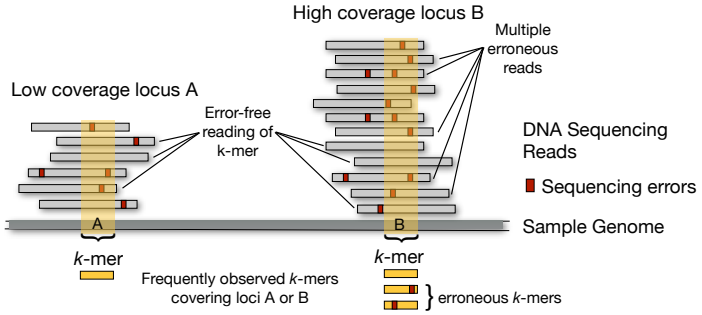


Figure 6.3: Locus A is covered by several reads that contain sequencing errors. However, not all reads contain errors in the k-mer covering this genome region and less than two or more of the reads covering the locus contain the same sequencing error. In contrast locus B is covered by many sequence reads due to to MDA-derived coverage bias and the same sequencing error is observed several times. Consequently, additional, erroneous k-mers are observed for this genome region.

of the reads, we expect partial reads to contain fewer erroneous k-mers while still containing a large proportion of the genomic ones. Note that if this turns out to be the case, partial reads would be enough to at least identify a pathogen, if not fully characterize it.

6.2.3 Preliminary results

To get an understanding of what proportion of the genomic k-mers may be recovered from partial reads and how the rate of false frequent k-mers increases with partial read length, we performed the following experiments on two bacterial (*E.coli* 760.87x and *Salmonella bongori* 107x average genome coverage) and one eukaryotic (the diatom *T. pseudonana* 85x genome coverage) MDA-amplified Illumina datasets. We used sequencing libraries from biological samples, and simulated only the purely technical step of producing partial reads. That is, we extracted the first 25, 35, 45, ... nucleotides of all reads and identified the k-mers that appeared at least twice in the partial reads. Since reference genomes were available for all the datasets, we were able to identify the number of genomic and erroneous k-mers. We found that (see Figure 6.4) for the high coverage *E. coli* dataset (760.87-fold coverage), we recover 96.76% of all true 31-mers from partial reads of length 45 nt, and only 27.46% of 31-mers appearing twice or more are erroneous or False Frequent (FF). Similarly, for *S. bongori* we recovered 97.37% of the true 31-mers from partial reads of length 39 nt, with a very low proportion of FF k-mers of 9.97%. As the *T. pseudonana* dataset has a comparatively low coverage we consider partial reads of length 55 nt from which we recover 97.89% of the true 31-mers with a proportion of FF k-mers of 23.93%. This behavior is consistently observed for other k-mer lengths (data not presented).

This clearly indicates that for a fixed k-mer length it is possible to extract almost all true frequent k-mers from partial reads which have a length less than twice the k-mer length. Even more striking, as partial read length increases, the very small missing percentage ($< 3\%$) of true frequent k-mers will be observed only at an enormous cost of false observations: the proportion of FF k-mers in the set of frequent k-mers increases dramatically, reaching 87.17% for *E. coli*, 35.48% for *S. bongori* and 43.50%

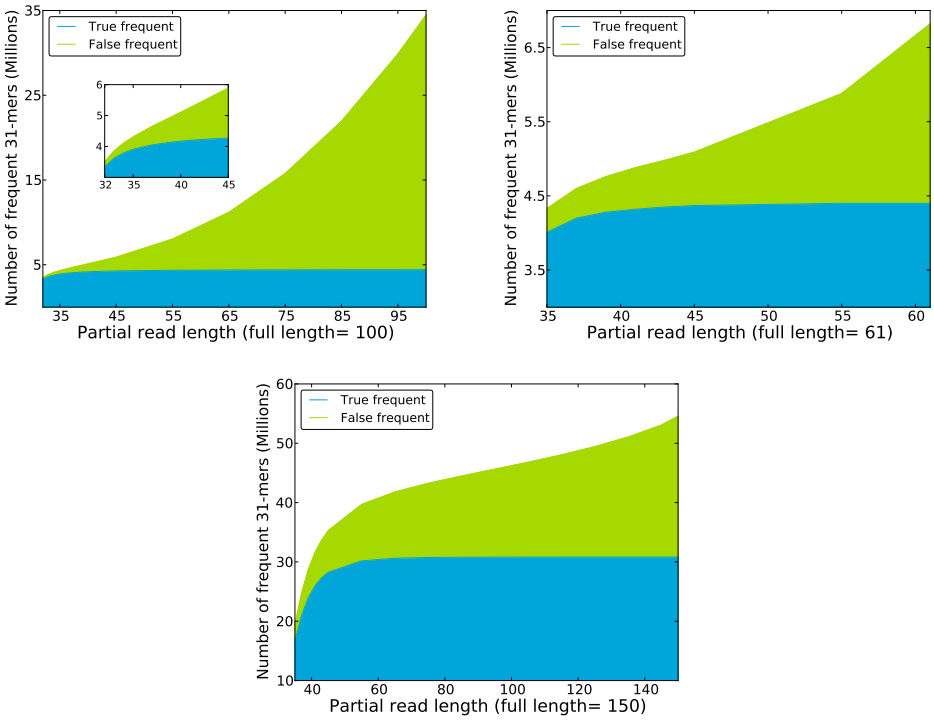


Figure 6.4: The number of true frequent 31-mers (31-mers that are observed in the genome, blue) and false frequent (green) 31-mers observed as a function of increasing partial read length for two bacterial (*E. coli*, coverage 760.87x (top left) and *Salmonella bongori*, coverage 107x, (top right)) and one eukaryotic (*T. pseudonana*, coverage 85x(bottom)) MDA amplified Illumina datasets. Note that the blue curve attains its maximal value rapidly while the green curve continues to grow at a speed depending on the coverage and other variables.

for *T. pseudonana* at full read length.

Specialized single cell assemblers like SPAdes and IBDB-UD (Peng *et al.*, 2012) perform assembly in iterative mode, i.e., in stages with increasing k-mer lengths (e.g., 21, 31, 41, ..., 127). Using this approach, shorter k-mer lengths maintain contiguity at low coverage, whereas longer k-mers provide best possible repeat resolution at later stages. We followed a similar strategy for assembly. We implemented an iterative assembly software using ABySS as the assembler during individual iterations. Since a large proportion of the genomic k-mers are available in partial reads, the iterative assembly stage with k-mer length k can commence as soon as partial reads of length slightly larger than k are available. We observed that similar to extracting true k-mers, the relevant statistics of assembly quality such as contig length (a contig is an assembled, contiguous fragment of the sequenced genome) increased rapidly, see Table 6.1. In Wooley *et al.* (2010), Table 2, the authors notes that a sequence length of 1,000-5,000 is expected to contain information about multi-domain genes while a sequence length of 5,000-10,000 is needed for identifying longer operons and cis-control elements. As Table 6.1 shows, even with a partial read length of 25 nt, the average contig length is more than 10,000 and indeed $\approx 80\%$ of the predicted proteins had significant homologs. Also note that over 83% of predicted proteins with annotated homologs were found with partial reads of length 55 nt, in other words, with about half of the sequencing completed. The high proportion of mapped contigs clearly indicates that this strategy is successful at avoiding incorrect assemblies. Curiously, the assembly quality degrades for full length reads, which is likely caused by an increased fraction of false frequent k-mers in the k-mer spectrum. This indicates that better control of the proportion of false frequent k-mers will improve assembly in general.

Table 6.1: Quality measures of the iterative assembly of a MDA amplified Illumina library shows that even early iterations of the assembly provide sufficiently large contigs for downstream analysis. Contigs were produced using `ABySS` from the paired partial (short) reads and contigs from the previous assembly stage (considered long reads). A contig having an alignment of length $\geq 90\%$ of its own length to the reference genome is considered mapped. Proteins were predicted using `GeneMark.hmm` (Lukashin and Borodovsky, 1998). Again, a protein having an alignment of length $\geq 70\%$ of its own length to a reference protein is considered to have an annotated homolog. Both contigs and proteins were aligned using `Blast` (Altschul *et al.*, 1990).

Length (bp)		**Contigs**		**N50 (bp)**	**Contig length (bp)**		**Assembly**	**Proteins**	
***k*-mer**	**Partial read**	**No.**	**Mapped**		**Maximal**	**Average**	**size (bp)**	**Predicted**	**w/ Homologs**
21	25	279	78.13%	56,502	198,314	12,057	3,364,177	3,637	2,894
31	35	203	56.15%	80,785	210,855	16,978	3,446,586	3,567	2,986
41	45	174	77.01%	82,593	210,871	19,894	3,461,679	3,526	2,978
51	55	170	79.41%	87,020	210,881	20,462	3,478,623	3,540	3,003
61	65	169	84.61%	87,030	210,891	20,698	3,498,110	3,539	3,017
63	75	141	82.27%	86,853	210,893	27,257	3,843,316	3,873	3,301
63	100 (full)	193	79.27%	67,732	178,496	22,409	4,324,941	4,523	3,592

6.3 Discussion

For all the preliminary experiments, we worked with datasets that had a reference genome. For unknown organisms, where we do not have any reference assembly, the challenge would be to predict a partial read length such that a significant fraction of the genomic k-mers appear as frequent while minimizing the rate of false frequent k-mers. Note that, in this case, we only observe the frequent k-mers and cannot identify the false frequent k-mers. However, we would like to have an estimate about how many genomic k-mers have been seen and what is the proportion of false frequent k-mers (False discovery rate). We propose to model this as a *coupon collecting* problem where coupons are allowed to be damaged (a genomic k-mer becomes erroneous) and we are only interested in coupons collected two or more times (the *Double Dixie cup* problem). An important component of this problem is the frequency distribution of the k-mers (coupons) being collected. Again, for multi-cell libraries where the genome coverage is nearly uniform, we may assume a uniform distribution for k-mer frequencies. For our model to be applicable to single cell libraries, we need a thorough understanding of the k-mer frequency distribution of WGA libraries. The k-mer frequency distribution for single-cell libraries are not well studied and understanding them is important not only for our purpose but will also provide an important pillar for single-cell genomics in general.

Genetics is a very potent tool for identifying and characterizing organisms. However, traditional genetics lacks the speed necessary to be of use for rapid pathogen detection. SCG promises to fill this gap and our proposal is to further speed it up with bioinformatic analysis that is performed concurrently with sequencing. Our preliminary results show that this is possible since partial reads may contain sufficient information for *in silico* analysis and identification of pathogens and understanding their pathogenicity via identification of genes. The challenge is to determine how much partial information is sufficient. Our model requires knowledge of the k-mer frequency distribution for WGA libraries. Currently this is an open problem and we believe, solving this will be an important contribution to Single Cell Genomics.

Chapter 7

Conclusion

In this thesis, I have presented my contributions regarding method development that aim to improve genome assembly and some applications of assembly to single cell genomics. We began with one of the most fundamental problems of bioinformatics: k-mer frequency counting. For large read libraries, computing k-mer frequencies can be a major computational challenge. In Chapter 3, we present a tool named Turtle (Roy *et al.*, 2014b) that achieves improved performance by implementing algorithms designed to reduce cache misses. We present a counter-intuitive approach of using $O(n \log n)$ algorithms for a problem that may be solved in $O(n)$. We argued and showed that an $O(n \log n)$ algorithm that performs localized memory access can out-perform an $O(n)$ algorithm that performs random memory access simply because of having far fewer cache misses. We also reduce memory requirements by using a Bloom filter to eliminate infrequent k-mers and storing frequent ones in an array instead of a hash. However, as sequencing library size increases, I believe future bioinformatic tools have to be designed with external memory algorithms. Deorowicz *et al.* (2013) has already demonstrated the potential of disk based external memory algorithms by building a k-mer counting tool called KMC. I believe this is the correct future direction for bioinformatic algorithm development.

Contig scaffolding is another classic problem in bioinformatics. Contig scaffolding is usually formulated as a graph traversal problem where contigs are the vertices and paired reads are represented as edges in the graph. An ideal contig graph should have a linear structure. However, due to sequencing errors, ambiguous read mapping and mis-assembly of contigs, the graph gets much more complicated in structure. Therefore, filtering unreliable pairs (and thus edges) from the contig graph is very important for

improving scaffolding. In chapter 4, we present a set of linear equations capable of filtering out such unreliable paired reads and also predicting the relative position and orientation of contigs connected with paired reads. This greatly simplifies the scaffolding problem and we show (Roy *et al.*, 2012) that a very naive scaffolding algorithm is capable of out-performing state of the art scaffolders. Our experiments also show that correct contigs produce a much cleaner contig graph and thereby further simplifies the scaffolding problem.

I also produced the first draft genome assembly of a single cell belonging to the broadly distributed group of MAST-4 marine stramenopiles (Roy *et al.*, 2014a). Using specialized assemblers, we assembled an MDA amplified single-cell library and predicted approximately 7,000 protein-encoding genes in the MAST-4 genome. With the identified protein-encoding genes, we were able to robustly position the marine organism into the ToL using multi-gene phylogenetics and gain insights into its metabolic capabilities. Our success demonstrates that Single cell genomics can be a powerful and effective tool for studying the vast majority of organisms that cannot be cultured in the lab.

My current and final project is rapid pathogen detection using SCG. An ideal diagnostic tool would identify the pathogen and also inform about its drug resistance in a timely manner so that appropriate therapy could begin as soon as possible. Traditional clinical methods such as staining and microscopy, culturing can take an extended period of time while still producing ambiguous results. The most successful rapid pathogen diagnostic tools currently available are based on genetic markers. Though some of them (e.g. PCR/ESI-MS) are very efficient at identifying a pathogen using markers such as 16s rDNA and housekeeping genes, they are very limited in terms of identifying drug-resistance and providing a wholesome picture about the capabilities of the pathogen. Whole genome sequencing provides an opportunity to overcome this but the traditional techniques are limited due to culturing to produce enough genetic material. SCG is capable of amplifying a very minute amount of genetic material to an amount that is sufficient for further genetic analysis. However, the amplification process introduces large amount of coverage variation in the data which makes sequence analysis difficult. We are currently working to identify reliable genetic information from amplified

single-cell libraries in order to make sequence analysis feasible. We are developing novel models for understanding the power of partial reads for performing sequence analysis in parallel with sequencing to increase the speed of sequence analysis and ultimately pathogen detection. This project also aims to characterize the coverage bias in MDA which, if successful, will be an important contribution to single cell genomics. The ability to genetically analyze and understand any organism's capabilities merely from sequence analysis of an environmental sample is the ultimate goal and I am thankful to have the opportunity to participate in this endeavor. The community is vibrantly trying to improve both data generation (WGA) and analysis (sequence analysis of WGA libraries) and I believe, success is only a matter of time.

Appendices

Appendix A

Abbreviations

Abbreviations

BF Bloom filter.

DNA Deoxyribonucleic acid.

EST Expressed sequence tag.

HGT Horizontal gene transfer.

HTS High Throughput Sequencing.

MDA Multiple Displacement Amplification.

MG Metagenomics.

MS Mass spectrometry.

NGS Next Generation Sequencing.

PCR Polymerase chain reaction.

PRs Paired reads.

SCG Single Cell Genomics.

TB Tuberculosis.

ToL Tree of life.

WGA Whole Genome Amplification.

WGS Whole Genome Sequencing.

Appendix B

Running commands for the k-mer counting tools compared in Chapter 3

In this section, we present the commands used for running the various k-mer counting tools to produce the results presented in Tables 3.3, 3.5 and 3.6.

B.1 Jellyfish-1.1.11

Please note that the parameter `-s` was chosen to minimize the number of intermediate files generated while still fitting in memory as suggested by the Jellyfish manual.

```
jellyfish count -m 31 -C -s 270M -t num_threads DM.fasta
jellyfish count -m 31 -C -s 2700M -t num_threads GG.fasta
jellyfish count -m 31 -C -s 6G -t num_threads ZM.fasta
jellyfish count -m 31 -C -s 6G -t num_threads HS.fasta
```

The `merge` (where applicable) and `dump` commands are:

```
jellyfish merge mer_counts_*
jellyfish dump -L 2 -o jf_kmers.fa mer_counts_merged.jf
```

B.2 Khmer-0.7.1

```
python load-into-counting.py -b -k 31 -x 5e9 -T num_threads out.kh DM.fasta
python load-into-counting.py -b -k 31 -x 16e9 -T num_threads out.kh GG.fasta
python load-into-counting.py -b -k 31 -x 64e9 -T num_threads out.kh ZM.fasta
python load-into-counting.py -b -k 31 -x 64e9 -T num_threads out.kh HS.fasta
```

B.3 KMC 0.3

Please note that unlike the other tools, KMC does not allow one parameter for specifying the number of threads to use. Instead there are separate parameters for choosing the number of compacting threads (-sc), number of FASTQ reading threads (-sf), number of splitting threads (-sp), number of sorter threads (-sr) and number of threads per single sorter (-so). KMC 0.3 was run using the following parameters for the various number of threads:

5 threads:

```
kmc -k31 -cs65536 -sf1 -sc1 -sr3 -sp1 -so1 -mX -fa read_library.fasta count_dump
```

7 threads:

```
kmc -k31 -cs65536 -sf1 -sc1 -sr4 -sp2 -so1 -mX -fa read_library.fasta count_dump
```

9 threads:

```
kmc -k31 -cs65536 -sf1 -sc2 -sr5 -sp2 -so1 -mX -fa read_library.fasta count_dump
```

11 threads:

```
kmc -k31 -cs65536 -sf1 -sc2 -sr7 -sp2 -so1 -mX -fa read_library.fasta count_dump
```

13 threads:

```
kmc -k31 -cs65536 -sf1 -sc2 -sr8 -sp3 -so1 -mX -fa read_library.fasta count_dump
```

15 threads:

```
kmc -k31 -cs65536 -sf1 -sc3 -sr9 -sp3 -so1 -mX -fa read_library.fasta count_dump
```

17 threads:

```
kmc -k31 -cs65536 -sf1 -sc3 -sr9 -sp4 -so1 -mX -fa read_library.fasta count_dump
```

19 threads:

```
kmc -k31 -cs65536 -sf1 -sc5 -sr10 -sp4 -so1 -mX -fa read_library.fasta count_dump
```

The `-m` parameter was set to 6, 47, 82, 109 for the DM, GG, ZM, HS libraries respectively. The `dump` command was run as follows:

```
kmc_dump count_dump kmer_count.fa
```

B.4 scTurtle-0.1

```
scTurtle32 -i DM.fasta -o kmer_counts -k 31 -n 135000000 -t num_threads
scTurtle32 -i GG.fasta -o kmer_counts -k 31 -n 1150000000 -t num_threads
scTurtle32 -i ZM.fasta -o kmer_counts -k 31 -n 2100000000 -t num_threads
scTurtle32 -i HS.fasta -o kmer_counts -k 31 -n 2800000000 -t num_threads
```

B.5 cTurtle-0.1

```
cTurtle32 -i DM.fasta -o kmer_counts -k 31 -n 135000000 -t num_threads
cTurtle32 -i GG.fasta -o kmer_counts -k 31 -n 1150000000 -t num_threads
cTurtle32 -i ZM.fasta -o kmer_counts -k 31 -n 2100000000 -t num_threads
cTurtle32 -i HS.fasta -o kmer_counts -k 31 -n 2800000000 -t num_threads
```

Please note that the results presented in Table 5 was also run using the same parameters except for `-k`, the k-mer size.

B.6 BFCounter-0.2

```
BFCounter count -k 31 -n 135000000 -o 31_mer.bfc DM.fastq
BFCounter count -k 31 -n 1150000000 -o 31_mer.bfc GG.fastq
BFCounter count -k 31 -n 2100000000 -o 31_mer.bfc ZM.fastq
BFCounter count -k 31 -n 2800000000 -o 31_mer.bfc HS.fastq
```

The `dump` command was run as follows:

```
BFCounter dump -k 31 -i 31_mer.bfc -o 31_bfc.dump
```

B.7 DSK-1.4811

```
dsk DM.fasta 31 -t 2 -m 5000
dsk GG.fasta 31 -t 2 -m 50000
dsk ZM.fasta 31 -t 2 -m 80000
```

```
dsk HS.fasta 31 -t 2 -m 100000
```

Appendix C

Acknowledgement of previous publications

A large part of the work described in this thesis was previously published in Roy *et al.* (2012), Roy *et al.* (2014b) and Roy *et al.* (2014a).

Bibliography

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, **215**(3), 403–410.

Andersson, J. O. (2009). Gene transfer and diversification of microbial eukaryotes. *Annu Rev Microbiol*, **63**, 177–193.

Archibald, J. M., Rogers, M. B., Toop, M., Ishida, K.-I., and Keeling, P. J. (2003). Lateral gene transfer and the evolution of plastid-targeted proteins in the secondary plastid-containing alga *bigelowiella natans*. *Proc Natl Acad Sci U S A*, **100**(13), 7678–7683.

Armbrust, E. V., Berges, J. A., Bowler, C., Green, B. R., Martinez, D., Putnam, N. H., Zhou, S., Allen, A. E., Apt, K. E., Bechner, M., Brzezinski, M. A., Chaal, B. K., Chiovitti, A., Davis, A. K., Demarest, M. S., Detter, J. C., Glavina, T., Goodstein, D., Hadi, M. Z., Hellsten, U., Hildebrand, M., Jenkins, B. D., Jurka, J., Kapitonov, V. V., Kr?ger, N., Lau, W. W. Y., Lane, T. W., Larimer, F. W., Lippmeier, J. C., Lucas, S., Medina, M., Montsant, A., Obornik, M., Parker, M. S., Palenik, B., Pazour, G. J., Richardson, P. M., Rynearson, T. A., Saito, M. A., Schwartz, D. C., Thamatrakoln, K., Valentin, K., Vardi, A., Wilkerson, F. P., and Rokhsar, D. S. (2004). The genome of the diatom *thalassiosira pseudonana*: ecology, evolution, and metabolism. *Science*, **306**(5693), 79–86.

Avery, O. T., MacLeod, C. M., and McCarty, M. (1944). Studies on the chemical nature of the substance inducing transformation of pneumococcal types. induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type iii. *Journal of Experimental Medicine*, (4).

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., and Pevzner, P. A. (2012). Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol*, **19**(5), 455–477.

Batzoglou, S., Jaffe, D. B., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J. P., and Lander, E. S. (2002). Arachne: a whole-genome shotgun assembler. *Genome Res*, **12**(1), 177–189.

Behnke, A., Engel, M., Christen, R., Nebel, M., Klein, R. R., and Stoeck, T. (2011). Depicting more accurate pictures of protistan community complexity using pyrosequencing of hypervariable ssu rrna gene regions. *Environmental microbiology*, **13**(2), 340–349.

Bender, M. A., Demaine, E. D., and Farach-Colton, M. (2005). Cache-oblivious b-trees. *SIAM J. Comput.*, **35**(2), 341–358.

Benjamini, Y. and Speed, T. P. (2012). Summarizing and correcting the gc content bias in high-throughput sequencing. *Nucleic Acids Res*, **40**(10), e72.

Bhattacharya, D., Price, D. C., Yoon, H. S., Yang, E. C., Poulton, N. J., Andersen, R. A., and Das, S. P. (2012). Single cell genome analysis supports a link between phagotrophy and primary plastid endosymbiosis. *Sci Rep*, **2**, 356.

Bhattacharya, D., Price, D. C., Chan, C. X., Qiu, H., Rose, N., Ball, S., Weber, A. P. M., Arias, M. C., Henrissat, B., Coutinho, P. M., Krishnan, A., Z?uner, S., Morath, S., Hilliou, F., Egizi, A., Perrineau, M.-M., and Yoon, H. S. (2013). Genome of the red alga *porphyridium purpureum. Nat Commun*, **4**, 1941.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, **13**(7), 422–426.

Boetzer, M., Henkel, C. V., Jansen, H. J., Butler, D., and Pirovano, W. (2011). Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, **27**(4), 578–579.

Brown, T. (2002). *Genomes: Chapter2 Genome Anatomies*. Oxford: Wiley-Liss.

Burge, C. and Karlin, S. (1997). Prediction of complete gene structures in human genomic dna. *J Mol Biol*, **268**(1), 78–94.

Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I. A., Belmonte, M. K., Lander, E. S., Nusbaum, C., and Jaffe, D. B. (2008). Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Res*, **18**(5), 810–820.

Capella-Gutierrez, S., Kauff, F., and Gabaldn, T. (2014). A phylogenomics approach for selecting robust sets of phylogenetic markers. *Nucleic Acids Res*, **42**(7), e54.

Chan, C. X., Soares, M. B., Bonaldo, M. F., Wisecaver, J. H., Hackett, J. D., Anderson, D. M., Erdner, D. L., and Bhattacharya, D. (2012). Analysis of *alexandrium tamarense* (dinophyceae) genes reveals the complex evolutionary history of a microbial eukaryote. *J Phycol*, **48**(5), 1130–1142.

Chapman, J. A., Ho, I., Sunkara, S., Luo, S., Schroth, G. P., and Rokhsar, D. S. (2011). Meraculous: de novo genome assembly with short paired-end reads. *PLoS One*, **6**(8), e23501.

Chitsaz, H., Yee-Greenbaum, J. L., Tesler, G., Lombardo, M.-J., Dupont, C. L., Badger, J. H., Novotny, M., Rusch, D. B., Fraser, L. J., Gormley, N. A., Schulz-Trieglaff, O., Smith, G. P., Evers, D. J., Pevzner, P. A., and Lasken, R. S. (2011). Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. *Nat Biotechnol*, **29**(10), 915–921.

Consortium, M. G. S., Waterston, R. H., Lindblad-Toh, K., Birney, E., Rogers, J., Abril, J. F., Agarwal, P., Agarwala, R., Ainscough, R., Alexandersson, M., An,

P., Antonarakis, S. E., Attwood, J., Baertsch, R., Bailey, J., Barlow, K., Beck, S., Berry, E., Birren, B., Bloom, T., Bork, P., Botcherby, M., Bray, N., Brent, M. R., Brown, D. G., Brown, S. D., Bult, C., Burton, J., Butler, J., Campbell, R. D., Carninci, P., Cawley, S., Chiaromonte, F., Chinwalla, A. T., Church, D. M., Clamp, M., Clee, C., Collins, F. S., Cook, L. L., Copley, R. R., Coulson, A., Couronne, O., Cuff, J., Curwen, V., Cutts, T., Daly, M., David, R., Davies, J., Delehaunty, K. D., Deri, J., Dermitzakis, E. T., Dewey, C., Dickens, N. J., Diekhans, M., Dodge, S., Dubchak, I., Dunn, D. M., Eddy, S. R., Elnitski, L., Emes, R. D., Eswara, P., Eyras, E., Felsenfeld, A., Fewell, G. A., Flicek, P., Foley, K., Frankel, W. N., Fulton, L. A., Fulton, R. S., Furey, T. S., Gage, D., Gibbs, R. A., Glusman, G., Gnerre, S., Goldman, N., Goodstadt, L., Grafham, D., Graves, T. A., Green, E. D., Gregory, S., Guig, R., Guyer, M., Hardison, R. C., Haussler, D., Hayashizaki, Y., Hillier, L. W., Hinrichs, A., Hlavina, W., Holzer, T., Hsu, F., Hua, A., Hubbard, T., Hunt, A., Jackson, I., Jaffe, D. B., Johnson, L. S., Jones, M., Jones, T. A., Joy, A., Kamal, M., Karlsson, E. K., Karolchik, D., Kasprzyk, A., Kawai, J., Keibler, E., Kells, C., Kent, W. J., Kirby, A., Kolbe, D. L., Korf, I., Kucherlapati, R. S., Kulbokas, E. J., Kulp, D., Landers, T., Leger, J. P., Leonard, S., Letunic, I., Levine, R., Li, J., Li, M., Lloyd, C., Lucas, S., Ma, B., Maglott, D. R., Mardis, E. R., Matthews, L., Mauceli, E., Mayer, J. H., McCarthy, M., McCombie, W. R., McLaren, S., McLay, K., McPherson, J. D., Meldrim, J., Meredith, B., Mesirov, J. P., Miller, W., Miner, T. L., Mongin, E., Montgomery, K. T., Morgan, M., Mott, R., Mullikin, J. C., Muzny, D. M., Nash, W. E., Nelson, J. O., Nhan, M. N., Nicol, R., Ning, Z., Nusbaum, C., O'Connor, M. J., Okazaki, Y., Oliver, K., Overton-Larty, E., Pachter, L., Parra, G., Pepin, K. H., Peterson, J., Pevzner, P., Plumb, R., Pohl, C. S., Poliakov, A., Ponce, T. C., Ponting, C. P., Potter, S., Quail, M., Reymond, A., Roe, B. A., Roskin, K. M., Rubin, E. M., Rust, A. G., Santos, R., Sapojnikov, V., Schultz, B., Schultz, J., Schwartz, M. S., Schwartz, S., Scott, C., Seaman, S., Searle, S., Sharpe, T., Sheridan, A., Shownkeen, R., Sims, S., Singer, J. B., Slater, G., Smit, A., Smith, D. R., Spencer, B., Stabenau, A., Stange-Thomann, N., Sugnet, C., Suyama, M., Tesler, G., Thompson, J., Torrents, D., Trevaskis, E., Tromp, J., Ucla, C., Ureta-Vidal, A., Vinson, J. P., Niederhausern, A. C. V., Wade, C. M., Wall, M., Weber, R. J., Weiss, R. B., Wendl, M. C., West, A. P., Wetterstrand, K., Wheeler, R., Whelan, S., Wierzbowski, J., Willey, D., Williams, S., Wilson, R. K., Winter, E., Worley, K. C., Wyman, D., Yang, S., Yang, S.-P., Zdobnov, E. M., Zody, M. C., and Lander, E. S. (2002). Initial sequencing and comparative analysis of the mouse genome. *Nature*, **420**(6915), 520–562.

Crick, F. H., Barnett, L., Brenner, S., and Watts-Tobin, R. J. (1961). General nature of the genetic code for proteins. *Nature*, **192**, 1227–1232.

Curtis, B. A., Tanifuji, G., Burki, F., Gruber, A., Irimia, M., Maruyama, S., Arias, M. C., Ball, S. G., Gile, G. H., Hirakawa, Y., Hopkins, J. F., Kuo, A., Rensing, S. A., Schmutz, J., Symeonidi, A., Elias, M., Eveleigh, R. J. M., Herman, E. K., Klute, M. J., Nakayama, T., Oborn?k, M., Reyes-Prieto, A., Armbrust, E. V., Aves, S. J., Beiko, R. G., Coutinho, P., Dacks, J. B., Durnford, D. G., Fast, N. M., Green, B. R., Grisdale, C. J., Hempel, F., Henrissat, B., H?ppner, M. P., Ishida, K.-I., Kim, E.,

Ko?en?, L., Kroth, P. G., Liu, Y., Malik, S.-B., Maier, U. G., McRose, D., Mock, T., Neilson, J. A. D., Onodera, N. T., Poole, A. M., Pritham, E. J., Richards, T. A., Rocap, G., Roy, S. W., Sarai, C., Schaack, S., Shirato, S., Slamovits, C. H., Spencer, D. F., Suzuki, S., Worden, A. Z., Zauner, S., Barry, K., Bell, C., Bharti, A. K., Crow, J. A., Grimwood, J., Kramer, R., Lindquist, E., Lucas, S., Salamov, A., McFadden, G. I., Lane, C. E., Keeling, P. J., Gray, M. W., Grigoriev, I. V., and Archibald, J. M. (2012). Algal genomes reveal evolutionary mosaicism and the fate of nucleomorphs. *Nature*, **492**(7427), 59–65.

Dayarian, A., Michael, T. P., and Sengupta, A. M. (2010). Sopra: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, **11**, 345.

Dean, F. B., Hosono, S., Fang, L., Wu, X., Faruqi, A. F., Bray-Ward, P., Sun, Z., Zong, Q., Du, Y., Du, J., *et al.* (2002). Comprehensive human genome amplification using multiple displacement amplification. *Proceedings of the National Academy of Sciences*, **99**(8), 5261–5266.

Dellinger, R. P., Levy, M. M., Carlet, J. M., Bion, J., Parker, M. M., Jaeschke, R., Reinhart, K., Angus, D. C., Brun-Buisson, C., Beale, R., Calandra, T., Dhainaut, J.-F., Gerlach, H., Harvey, M., Marini, J. J., Marshall, J., Ranieri, M., Ramsay, G., Sevransky, J., Thompson, B. T., Townsend, S., Vender, J. S., Zimmerman, J. L., and Vincent, J.-L. (2008). Surviving sepsis campaign: international guidelines for management of severe sepsis and septic shock: 2008. *Intensive Care Med*, **34**(1), 17–60.

Deorowicz, S., Debudaj-Grabysz, A., and Grabowski, S. (2013). Disk-based k-mer counting on a PC. *BMC Bioinformatics*, **14**, 160.

Donmez, N. (2012). *Polymorphism and Genome Assembly*. Ph.D. thesis, Department of Computer Science, University of Toronto.

Donmez, N. and Brudno, M. (2013). Scarpa: scaffolding reads with practical algorithms. *Bioinformatics*, **29**(4), 428–434.

Ecker, D. J., Sampath, R., Blyn, L. B., Eshoo, M. W., Ivy, C., Ecker, J. A., Libby, B., Samant, V., Sannes-Lowery, K. A., Melton, R. E., Russell, K., Freed, N., Barrozo, C., Wu, J., Rudnick, K., Desai, A., Moradi, E., Knize, D. J., Robbins, D. W., Hannis, J. C., Harrell, P. M., Massire, C., Hall, T. A., Jiang, Y., Ranken, R., Drader, J. J., White, N., McNeil, J. A., Crooke, S. T., and Hofstadler, S. A. (2005). Rapid identification and strain-typing of respiratory pathogens for epidemic surveillance. *Proc Natl Acad Sci U S A*, **102**(22), 8012–8017.

Ecker, D. J., Sampath, R., Li, H., Massire, C., Matthews, H. E., Toleno, D., Hall, T. A., Blyn, L. B., Eshoo, M. W., Ranken, R., Hofstadler, S. A., and Tang, Y.-W. (2010). New technology for rapid molecular diagnosis of bloodstream infections. *Expert Rev Mol Diagn*, **10**(4), 399–415.

Edgar, R. C. (2004). Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, **32**(5), 1792–1797.

English, A. C., Richards, S., Han, Y., Wang, M., Vee, V., Qu, J., Qin, X., Muzny, D. M., Reid, J. G., Worley, K. C., and Gibbs, R. A. (2012). Mind the gap: upgrading genomes with pacific biosciences rs long-read sequencing technology. *PLoS One*, **7**(11), e47768.

Evans, V. C., Barker, G., Heesom, K. J., Fan, J., Bessant, C., and Matthews, D. A. (2012). De novo derivation of proteomes from transcriptomes for transcript and protein identification. *Nat Methods*, **9**(12), 1207–1211.

Fan, L., Cao, P., Almeida, J., and Broder, A. Z. (2000). Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, **8**(3), 281–293.

Farrer, R. A., Kemen, E., Jones, J. D. G., and Studholme, D. J. (2009). De novo assembly of the pseudomonas syringae pv. syringae b728a genome using Illumina/Solexa short sequence reads. *FEMS Microbiol Lett*, **291**(1), 103–111.

Frommolt, R., Werner, S., Paulsen, H., Goss, R., Wilhelm, C., Zauner, S., Maier, U. G., Grossman, A. R., Bhattacharya, D., and Lohr, M. (2008). Ancient recruitment by chromists of green algal genes encoding enzymes for carotenoid biosynthesis. *Mol Biol Evol*, **25**(12), 2653–2667.

Gao, S., Nagarajan, N., and kin Sung, W. (2011). Opera: Reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *LNBI*, **6577**, 437–451.

Garey, Michael R.and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

Gilles, A., Meglcz, E., Pech, N., Ferreira, S., Malausa, T., and Martin, J.-F. (2011). Accuracy and quality assessment of 454 gs-flx titanium pyrosequencing. *BMC Genomics*, **12**, 245.

Gnerre, S., Maccallum, I., Przybylski, D., Ribeiro, F. J., Burton, J. N., Walker, B. J., Sharpe, T., Hall, G., Shea, T. P., Sykes, S., Berlin, A. M., Aird, D., Costello, M., Daza, R., Williams, L., Nicol, R., Gnirke, A., Nusbaum, C., Lander, E. S., and Jaffe, D. B. (2011). High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci U S A*, **108**(4), 1513–1518.

Guiry, M. D. (2012). How many species of algae are there? *Journal of Phycology*, **48**(5), 1057–1063.

Haubold, B. and Wiehe, T. (2006). How repetitive are genomes? *BMC Bioinformatics*, **7**, 541.

Hoff, K. J. (2009). The effect of sequencing errors on metagenomic gene prediction. *BMC Genomics*, **10**, 520.

Hopcroft, J. and Tarjan, R. (1973). Efficient algorithms for graph manipulation. *Communications of the ACM*, **16**, 372–378.

Huson, D. H., Reinert, K., and Myers, E. (2002). The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM*, **49**, 603–615.

Jaffe, D. B., Butler, J., Gnerre, S., Mauceli, E., Lindblad-Toh, K., Mesirov, J. P., Zody, M. C., and Lander, E. S. (2003). Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome Res*, **13**(1), 91–96.

Jain, R., Rivera, M. C., and Lake, J. A. (1999). Horizontal gene transfer among genomes: the complexity hypothesis. *Proc Natl Acad Sci U S A*, **96**(7), 3801–3806.

Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, **4**, 77–84.

Kanehisa, M. and Goto, S. (2000). Kegg: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, **28**(1), 27–30.

Kececioglu, J. D. and Myers, E. W. (1993). Combinatorial algorithms for dna sequence assembly. *Algorithmica*, **13**, 7–51.

Keeling, P. J. and Palmer, J. D. (2008). Horizontal gene transfer in eukaryotic evolution. *Nat Rev Genet*, **9**(8), 605–618.

Kelley, D. R., Schatz, M. C., and Salzberg, S. L. (2010). Quake: quality-aware detection and correction of sequencing errors. *Genome Biol*, **11**(11), R116.

Koren, S., Treangen, T. J., and Pop, M. (2011). Bambus 2: scaffolding metagenomes. *Bioinformatics*, **27**(21), 2964–2971.

Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., McKernan, K., Meldrim, J., Mesirov, J. P., Miranda, C., Morris, W., Naylor, J., Raymond, C., Rosetti, M., Santos, R., Sheridan, A., Sougnez, C., Stange-Thomann, N., Stojanovic, N., Subramanian, A., Wyman, D., Rogers, J., Sulston, J., Ainscough, R., Beck, S., Bentley, D., Burton, J., Clee, C., Carter, N., Coulson, A., Deadman, R., Deloukas, P., Dunham, A., Dunham, I., Durbin, R., French, L., Grafham, D., Gregory, S., Hubbard, T., Humphray, S., Hunt, A., Jones, M., Lloyd, C., McMurray, A., Matthews, L., Mercer, S., Milne, S., Mullikin, J. C., Mungall, A., Plumb, R., Ross, M., Shownkeen, R., Sims, S., Waterston, R. H., Wilson, R. K., Hillier, L. W., McPherson, J. D., Marra, M. A., Mardis, E. R., Fulton, L. A., Chinwalla, A. T., Pepin, K. H., Gish, W. R., Chissoe, S. L., Wendl, M. C., Delehaunty, K. D., Miner, T. L., Delehaunty, A., Kramer, J. B., Cook, L. L., Fulton, R. S., Johnson, D. L., Minx, P. J., Clifton, S. W., Hawkins, T., Branscomb, E., Predki, P., Richardson, P., Wenning, S., Slezak, T., Doggett, N., Cheng, J. F., Olsen, A., Lucas, S., Elkin, C., Uberbacher, E., Frazier, M., Gibbs, R. A., Muzny, D. M., Scherer, S. E., Bouck, J. B., Sodergren, E. J., Worley, K. C., Rives, C. M., Gorrell, J. H., Metzker, M. L., Naylor, S. L., Kucherlapati, R. S., Nelson, D. L., Weinstock, G. M., Sakaki, Y., Fujiyama, A., Hattori, M., Yada, T., Toyoda, A., Itoh, T., Kawagoe, C., Watanabe, H., Totoki, Y., Taylor, T., Weissenbach, J., Heilig, R., Saurin, W., Artiguenave, F., Brottier,

P., Bruls, T., Pelletier, E., Robert, C., Wincker, P., Smith, D. R., Doucette-Stamm, L., Rubenfield, M., Weinstock, K., Lee, H. M., Dubois, J., Rosenthal, A., Platzer, M., Nyakatura, G., Taudien, S., Rump, A., Yang, H., Yu, J., Wang, J., Huang, G., Gu, J., Hood, L., Rowen, L., Madan, A., Qin, S., Davis, R. W., Federspiel, N. A., Abola, A. P., Proctor, M. J., Myers, R. M., Schmutz, J., Dickson, M., Grimwood, J., Cox, D. R., Olson, M. V., Kaul, R., Raymond, C., Shimizu, N., Kawasaki, K., Minoshima, S., Evans, G. A., Athanasiou, M., Schultz, R., Roe, B. A., Chen, F., Pan, H., Ramser, J., Lehrach, H., Reinhardt, R., McCombie, W. R., de la Bastide, M., Dedhia, N., Blcker, H., Hornischer, K., Nordsiek, G., Agarwala, R., Aravind, L., Bailey, J. A., Bateman, A., Batzoglou, S., Birney, E., Bork, P., Brown, D. G., Burge, C. B., Cerutti, L., Chen, H. C., Church, D., Clamp, M., Copley, R. R., Doerks, T., Eddy, S. R., Eichler, E. E., Furey, T. S., Galagan, J., Gilbert, J. G., Harmon, C., Hayashizaki, Y., Haussler, D., Hermjakob, H., Hokamp, K., Jang, W., Johnson, L. S., Jones, T. A., Kasif, S., Kaspryzk, A., Kennedy, S., Kent, W. J., Kitts, P., Koonin, E. V., Korf, I., Kulp, D., Lancet, D., Lowe, T. M., McLysaght, A., Mikkelsen, T., Moran, J. V., Mulder, N., Pollara, V. J., Ponting, C. P., Schuler, G., Schultz, J., Slater, G., Smit, A. F., Stupka, E., Szustakowski, J., Thierry-Mieg, D., Thierry-Mieg, J., Wagner, L., Wallis, J., Wheeler, R., Williams, A., Wolf, Y. I., Wolfe, K. H., Yang, S. P., Yeh, R. F., Collins, F., Guyer, M. S., Peterson, J., Felsenfeld, A., Wetterstrand, K. A., Patrinos, A., Morgan, M. J., de Jong, P., Catanese, J. J., Osoegawa, K., Shizuya, H., Choi, S., Chen, Y. J., Szustakowki, J., and Consortium, I. H. G. S. (2001). Initial sequencing and analysis of the human genome. *Nature*, **409**(6822), 860–921.

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

Lasken, R. S. (2007). Single-cell genomic sequencing using multiple displacement amplification. *Curr Opin Microbiol*, **10**(5), 510–516.

Lepere, C., Demura, M., Kawachi, M., Romac, S., Probert, I., and Vaulot, D. (2011). Whole-genome amplification (wga) of marine photosynthetic eukaryote populations. *FEMS microbiology ecology*, **76**(3), 513–523.

Levinthal, D. (2008). Performance analysis guide for intel core i7 processor and intel xeon 5500 processors.

Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., and Wang, J. (2010a). De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res*, **20**(2), 265–272.

Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., and Wang, J. (2010b). De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res*, **20**(2), 265–272.

Li, R., Fan, W., Tian, G., Zhu, H., He, L., Cai, J., Huang, Q., Cai, Q., Li, B., Bai, Y., Zhang, Z., Zhang, Y., Wang, W., Li, J., Wei, F., Li, H., Jian, M., Li, J., Zhang, Z., Nielsen, R., Li, D., Gu, W., Yang, Z., Xuan, Z., Ryder, O. A., Leung, F. C.-C., Zhou, Y., Cao, J., Sun, X., Fu, Y., Fang, X., Guo, X., Wang, B., Hou, R., Shen, F., Mu, B., Ni, P., Lin, R., Qian, W., Wang, G., Yu, C., Nie, W., Wang, J., Wu, Z., Liang, H., Min, J., Wu, Q., Cheng, S., Ruan, J., Wang, M., Shi, Z., Wen, M., Liu, B., Ren, X., Zheng, H., Dong, D., Cook, K., Shan, G., Zhang, H., Kosiol, C., Xie, X., Lu, Z., Zheng, H., Li, Y., Steiner, C. C., Lam, T. T.-Y., Lin, S., Zhang, Q., Li, G., Tian, J., Gong, T., Liu, H., Zhang, D., Fang, L., Ye, C., Zhang, J., Hu, W., Xu, A., Ren, Y., Zhang, G., Bruford, M. W., Li, Q., Ma, L., Guo, Y., An, N., Hu, Y., Zheng, Y., Shi, Y., Li, Z., Liu, Q., Chen, Y., Zhao, J., Qu, N., Zhao, S., Tian, F., Wang, X., Wang, H., Xu, L., Liu, X., Vinar, T., Wang, Y., Lam, T.-W., Yiu, S.-M., Liu, S., Zhang, H., Li, D., Huang, Y., Wang, X., Yang, G., Jiang, Z., Wang, J., Qin, N., Li, L., Li, J., Bolund, L., Kristiansen, K., Wong, G. K.-S., Olson, M., Zhang, X., Li, S., Yang, H., Wang, J., and Wang, J. (2010c). The sequence and de novo assembly of the giant panda genome. *Nature*, **463**(7279), 311–317.

Lin, Y.-C., Campbell, T., Chung, C.-C., Gong, G.-C., Chiang, K.-P., and Worden, A. Z. (2012). Distribution patterns and phylogeny of marine stramenopiles in the north pacific ocean. *Appl Environ Microbiol*, **78**(9), 3387–3399.

Liong, M., Hoang, A. N., Chung, J., Gural, N., Ford, C. B., Min, C., Shah, R. R., Ahmad, R., Fernandez-Suarez, M., Fortune, S. M., Toner, M., Lee, H., and Weissleder, R. (2013). Magnetic barcode assay for genetic detection of pathogens. *Nat Commun*, **4**, 1752.

Liu, Y., Schrder, J., and Schmidt, B. (2012). Musket: a multistage k-mer spectrum based error corrector for illumina sequence data. *Bioinformatics*.

Lloyd, K. G., Schreiber, L., Petersen, D. G., Kjeldsen, K. U., Lever, M. A., Steen, A. D., Stepanauskas, R., Richter, M., Kleindienst, S., Lenk, S., Schramm, A., and J?rgensen, B. B. (2013). Predominant archaea in marine sediments degrade detrital proteins. *Nature*, **496**(7444), 215–218.

Lodish, H., Berk, A., Matsudaira, P., Kaiser, C., Krieger, M., Scott, M., Zipursky, S., and Darnell, J. (2004). *Molecular Cell Biology*. WH Freeman and Company, New York.

Lukashin, A. V. and Borodovsky, M. (1998). Genemark.hmm: new solutions for gene finding. *Nucleic Acids Res*, **26**(4), 1107–1115.

Macaulay, I. C. and Voet, T. (2014). Single cell genomics: advances and future perspectives. *PLoS Genet*, **10**(1), e1004126.

Marcais, G. and Kingsford, C. (2011). A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, **27**(6), 764–770.

Massana, R., Castresana, J., Balagu, V., Guillou, L., Romari, K., Groisillier, A., Valentin, K., and Pedras-Ali, C. (2004). Phylogenetic and ecological analysis of novel marine stramenopiles. *Appl Environ Microbiol*, **70**(6), 3528–3534.

Medvedev, P., Scott, E., Kakaradov, B., and Pevzner, P. (2011). Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, **27**(13), i137–i141.

Melsted, P. and Pritchard, J. K. (2011). Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics*, **12**, 333.

Miller, J. R., Delcher, A. L., Koren, S., Venter, E., Walenz, B. P., Brownley, A., Johnson, J., Li, K., Mobarry, C., and Sutton, G. (2008). Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**(24), 2818–2824.

Moustafa, A. and Bhattacharya, D. (2008). Phylosort: a user-friendly phylogenetic sorting tool and its application to estimating the cyanobacterial contribution to the nuclear genome of *chlamydomonas*. *BMC Evol Biol*, **8**, 6.

Moustafa, A., Beszteri, B., Maier, U. G., Bowler, C., Valentin, K., and Bhattacharya, D. (2009). Genomic footprints of a cryptic plastid endosymbiosis in diatoms. *Science*, **324**(5935), 1724–1726.

Parfrey, L. W., Grant, J., Tekle, Y. I., Lasek-Nesselquist, E., Morrison, H. G., Sogin, M. L., Patterson, D. J., and Katz, L. A. (2010). Broadly sampled multigene analyses yield a well-resolved eukaryotic tree of life. *Syst Biol*, **59**(5), 518–533.

Parra, G., Bradnam, K., and Korf, I. (2007). Cegma: a pipeline to accurately annotate core genes in eukaryotic genomes. *Bioinformatics*, **23**(9), 1061–1067.

Patterson, D. A. and Hennessey, J. L. (1998). *Computer Organization and Design: the Hardware/Software Interface, 2nd Edition*. Morgan Kaufmann Publishers, Inc., San Francisco, California.

Pawlowski, A., Jansson, M., Skld, M., Rottenberg, M. E., and Kllenius, G. (2012). Tuberculosis and hiv co-infection. *PLoS Pathog*, **8**(2), e1002464.

Pell, J., Hintze, A., Canino-Koning, R., Howe, A., Tiedje, J. M., and Brown, C. T. (2012). Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences*.

Peng, Y., Leung, H. C. M., Yiu, S. M., and Chin, F. Y. L. (2012). Idba-ud: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, **28**(11), 1420–1428.

Pevzner, P. A., Tang, H., and Waterman, M. S. (2001). An eulerian path approach to dna fragment assembly. *Proc Natl Acad Sci U S A*, **98**(17), 9748–9753.

Pop, M., Kosack, D. S., and Salzberg, S. L. (2004). Hierarchical scaffolding with Bambus. *Genome Res*, **14**(1), 149–159.

Price, D. C., Chan, C. X., Yoon, H. S., Yang, E. C., Qiu, H., Weber, A. P. M., Schwacke, R., Gross, J., Blouin, N. A., Lane, C., Reyes-Prieto, A., Durnford, D. G., Neilson, J. A. D., Lang, B. F., Burger, G., Steiner, J. M., L?ffelhardt, W., Meuser, J. E.,

Posewitz, M. C., Ball, S., Arias, M. C., Henrissat, B., Coutinho, P. M., Rensing, S. A., Symeonidi, A., Doddapaneni, H., Green, B. R., Rajah, V. D., Boore, J., and Bhattacharya, D. (2012). *Cyanophora paradoxa* genome elucidates origin of photosynthesis in algae and plants. *Science*, **335**(6070), 843–847.

Putze, F., Sanders, P., and Singler, J. (2010). Cache-, hash-, and space-efficient bloom filters. *J. Exp. Algorithmics*, **14**, 4:4.4–4:4.18.

Quail, M. A., Smith, M., Coupland, P., Otto, T. D., Harris, S. R., Connor, T. R., Bertoni, A., Swerdlow, H. P., and Gu, Y. (2012). A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC Genomics*, **13**, 341.

Raghunathan, A., Ferguson, H. R., Bornarth, C. J., Song, W., Driscoll, M., and Lasken, R. S. (2005). Genomic dna amplification from a single bacterium. *Applied and Environmental Microbiology*, **71**(6), 3342–3347.

Rizk, G., Lavenier, D., and Chikhi, R. (2013). Dsk: k-mer counting with very low memory usage. *Bioinformatics*, **29**(5), 652–653.

Rizzo, J. and Rouchka, E. C. (2007). Review of phylogenetic tree construction. *UNIVERSITY OF LOUISVILLE BIOINFORMATICS LABORATORY TECHNICAL REPORT SERIES*, **TR-ULBL-2007-01**.

Robbertse, B., Yoder, R. J., Boyd, A., Reeves, J., and Spatafora, J. W. (2011). Hal: an automated pipeline for phylogenetic analyses of genomic data. *PLoS Curr*, **3**, RRN1213.

Roch, S. (2006). A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, **3**(1), 92–.

Rodrigue, S., Malmstrom, R. R., Berlin, A. M., Birren, B. W., Henn, M. R., and Chisholm, S. W. (2009). Whole genome amplification and de novo assembly of single bacterial cells. *PLoS One*, **4**(9), e6864.

Rose, J. M., Caron, D. A., Sieracki, M. E., and Poulton, N. (2004). Counting heterotrophic nanoplanktonic protists in cultures and aquatic communities by flow cytometry. *Aquatic microbial ecology*, **34**(3), 263–277.

Roth, A., Schaberg, T., and Mauch, H. (1997). Molecular diagnosis of tuberculosis: current clinical validity and future perspectives. *European Respiratory Journal*, **10**(8), 1877–1891.

Roy, R. S., Chen, K. C., Sengupta, A. M., and Schliep, A. (2012). Sliq: simple linear inequalities for efficient contig scaffolding. *J Comput Biol*, **19**(10), 1162–1175.

Roy, R. S., Price, D. C., Schliep, A., Cai, G., Korobeynikov, A., Yoon, H. S., Yang, E. C., and Bhattacharya, D. (2014a). Single cell genome analysis of an uncultured heterotrophic stramenopile. *Sci Rep*, **4**, 4780.

Roy, R. S., Bhattacharya, D., and Schliep, A. (2014b). Turtle: Identifying frequent k-mers with cache-efficient algorithms. *Bioinformatics*.

Salmela, L., Mkinen, V., Vlimki, N., Ylinen, J., and Ukkonen, E. (2011). Fast scaffolding with small independent mixed integer programs. *Bioinformatics*.

Salomon, D. (1997). *Data Compression*. Springer.

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

Sboner, A., Mu, X. J., Greenbaum, D., Auerbach, R. K., and Gerstein, M. B. (2011). The real cost of sequencing: higher than you think! *Genome Biol*, **12**(8), 125.

Scally, A., Dutheil, J. Y., Hillier, L. W., Jordan, G. E., Goodhead, I., Herrero, J., Hobolth, A., Lappalainen, T., Mailund, T., Marques-Bonet, T., McCarthy, S., Montgomery, S. H., Schwalie, P. C., Tang, Y. A., Ward, M. C., Xue, Y., Yngvadottir, B., Alkan, C., Andersen, L. N., Ayub, Q., Ball, E. V., Beal, K., Bradley, B. J., Chen, Y., Clee, C. M., Fitzgerald, S., Graves, T. A., Gu, Y., Heath, P., Heger, A., Karakoc, E., Kolb-Kokocinski, A., Laird, G. K., Lunter, G., Meader, S., Mort, M., Mullikin, J. C., Munch, K., O'Connor, T. D., Phillips, A. D., Prado-Martinez, J., Rogers, A. S., Sajjadian, S., Schmidt, D., Shaw, K., Simpson, J. T., Stenson, P. D., Turner, D. J., Vigilant, L., Vilella, A. J., Whitener, W., Zhu, B., Cooper, D. N., de Jong, P., Dermitzakis, E. T., Eichler, E. E., Flicek, P., Goldman, N., Mundy, N. I., Ning, Z., Odom, D. T., Ponting, C. P., Quail, M. A., Ryder, O. A., Searle, S. M., Warren, W. C., Wilson, R. K., Schierup, M. H., Rogers, J., Tyler-Smith, C., and Durbin, R. (2012). Insights into hominid evolution from the gorilla genome sequence. *Nature*, **483**(7388), 169–175.

Simpson, J. T. (2012). *Efficient sequence assembly and variant calling using compressed data structures*. Ph.D. thesis, Queens College, Wellcome Trust Sanger Institute, University of Cambridge.

Simpson, J. T. and Durbin, R. (2012). Efficient de novo assembly of large genomes using compressed data structures. *Genome Res*, **22**(3), 549–556.

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J. M., and Birol, I. (2009). Abyss: a parallel assembler for short read sequence data. *Genome Res*, **19**(6), 1117–1123.

Sleator, R. D. (2010). An overview of the current status of eukaryote gene prediction strategies. *Gene*, **461**(1-2), 1–4.

Staden, R. (1979). A strategy of dna sequencing employing computer programs. *Nucleic Acids Res*, **6**(7), 2601–2610.

Stamatakis, A. (2006). Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, **22**(21), 2688–2690.

Stanke, M. and Morgenstern, B. (2005). Augustus: a web server for gene prediction in eukaryotes that allows user-defined constraints. *Nucleic Acids Res*, **33**(Web Server issue), W465–W467.

Stanke, M. and Waack, S. (2003). Gene prediction with a hidden markov model and a new intron submodel. *Bioinformatics*, **19 Suppl 2**, ii215–ii225.

Stepanauskas, R. (2012a). Single cell genomics: an individual look at microbes. *Current opinion in microbiology*, **15**(5), 613–620.

Stepanauskas, R. (2012b). Single cell genomics: an individual look at microbes. *Current Opinion in Microbiology*, **15**(5), 613 – 620.

Stepanauskas, R. and Sieracki, M. E. (2007). Matching phylogeny and metabolism in the uncultured marine bacteria, one cell at a time. *Proceedings of the National Academy of Sciences*, **104**(21), 9052–9057.

Struelens, M. J. (2010). Detection of microbial dnaemia: does it matter for sepsis management? *Intensive care medicine*, **36**(2), 193–195.

Talavera, G. and Castresana, J. (2007). Improvement of phylogenies after removing divergent and ambiguously aligned blocks from protein sequence alignments. *Syst Biol*, **56**(4), 564–577.

Townsend, J. P. (2007). Profiling phylogenetic informativeness. *Syst Biol*, **56**(2), 222–231.

van Belkum, A., Durand, G., Peyret, M., Chatellier, S., Zambardi, G., Schrenzel, J., Shortridge, D., Engelhardt, A., and Dunne, Jr, W. M. (2013). Rapid clinical bacteriology and its future impact. *Ann Lab Med*, **33**(1), 14–27.

Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., Gocayne, J. D., Amanatides, P., Ballew, R. M., Huson, D. H., Wortman, J. R., Zhang, Q., Kodira, C. D., Zheng, X. H., Chen, L., Skupski, M., Subramanian, G., Thomas, P. D., Zhang, J., Miklos, G. L. G., Nelson, C., Broder, S., Clark, A. G., Nadeau, J., McKusick, V. A., Zinder, N., Levine, A. J., Roberts, R. J., Simon, M., Slayman, C., Hunkapiller, M., Bolanos, R., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., Florea, L., Halpern, A., Hannenhalli, S., Kravitz, S., Levy, S., Mobarry, C., Reinert, K., Remington, K., Abu-Threideh, J., Beasley, E., Biddick, K., Bonazzi, V., Brandon, R., Cargill, M., Chandramouliswaran, I., Charlab, R., Chaturvedi, K., Deng, Z., Francesco, V. D., Dunn, P., Eilbeck, K., Evangelista, C., Gabrielian, A. E., Gan, W., Ge, W., Gong, F., Gu, Z., Guan, P., Heiman, T. J., Higgins, M. E., Ji, R. R., Ke, Z., Ketchum, K. A., Lai, Z., Lei, Y., Li, Z., Li, J., Liang, Y., Lin, X., Lu, F., Merkulov, G. V., Milshina, N., Moore, H. M., Naik, A. K., Narayan, V. A., Neelam, B., Nusskern, D., Rusch, D. B., Salzberg, S., Shao, W., Shue, B., Sun, J., Wang, Z., Wang, A., Wang, X., Wang, J., Wei, M., Wides, R., Xiao, C., Yan, C., Yao, A., Ye, J., Zhan, M., Zhang, W., Zhang, H., Zhao, Q., Zheng, L., Zhong, F., Zhong, W., Zhu, S., Zhao, S., Gilbert, D., Baumhueter, S., Spier, G., Carter, C., Cravchik, A., Woodage, T., Ali, F., An, H., Awe, A.,

Baldwin, D., Baden, H., Barnstead, M., Barrow, I., Beeson, K., Busam, D., Carver, A., Center, A., Cheng, M. L., Curry, L., Danaher, S., Davenport, L., Desilets, R., Dietz, S., Dodson, K., Doup, L., Ferriera, S., Garg, N., Gluecksmann, A., Hart, B., Haynes, J., Haynes, C., Heiner, C., Hladun, S., Hostin, D., Houck, J., Howland, T., Ibegwam, C., Johnson, J., Kalush, F., Kline, L., Koduru, S., Love, A., Mann, F., May, D., McCawley, S., McIntosh, T., McMullen, I., Moy, M., Moy, L., Murphy, B., Nelson, K., Pfannkoch, C., Pratts, E., Puri, V., Qureshi, H., Reardon, M., Rodriguez, R., Rogers, Y. H., Romblad, D., Ruhfel, B., Scott, R., Sitter, C., Smallwood, M., Stewart, E., Strong, R., Suh, E., Thomas, R., Tint, N. N., Tse, S., Vech, C., Wang, G., Wetter, J., Williams, S., Williams, M., Windsor, S., Winn-Deen, E., Wolfe, K., Zaveri, J., Zaveri, K., Abril, J. F., Guig, R., Campbell, M. J., Sjolander, K. V., Karlak, B., Kejariwal, A., Mi, H., Lazareva, B., Hatton, T., Narechania, A., Diemer, K., Muruganujan, A., Guo, N., Sato, S., Bafna, V., Istrail, S., Lippert, R., Schwartz, R., Walenz, B., Yooseph, S., Allen, D., Basu, A., Baxendale, J., Blick, L., Caminha, M., Carnes-Stine, J., Caulk, P., Chiang, Y. H., Coyne, M., Dahlke, C., Mays, A., Dombroski, M., Donnelly, M., Ely, D., Esparham, S., Fosler, C., Gire, H., Glanowski, S., Glasser, K., Glodek, A., Gorokhov, M., Graham, K., Gropman, B., Harris, M., Heil, J., Henderson, S., Hoover, J., Jennings, D., Jordan, C., Jordan, J., Kasha, J., Kagan, L., Kraft, C., Levitsky, A., Lewis, M., Liu, X., Lopez, J., Ma, D., Majoros, W., McDaniel, J., Murphy, S., Newman, M., Nguyen, T., Nguyen, N., Nodell, M., Pan, S., Peck, J., Peterson, M., Rowe, W., Sanders, R., Scott, J., Simpson, M., Smith, T., Sprague, A., Stockwell, T., Turner, R., Venter, E., Wang, M., Wen, M., Wu, D., Wu, M., Xia, A., Zandieh, A., and Zhu, X. (2001). The sequence of the human genome. *Science*, **291**(5507), 1304–1351.

Wang, Z., Chen, Y., and Li, Y. (2004). A brief review of computational gene prediction methods. *Genomics Proteomics Bioinformatics*, **2**(4), 216–221.

Warren, H. S. (2012). *Hackers Delight*. Addison-Wesley Professional, 2nd edition edition.

WATSON, J. D. and CRICK, F. H. (1953). The structure of DNA. *Cold Spring Harb Symp Quant Biol*, **18**, 123–131.

Wooley, J. C., Godzik, A., and Friedberg, I. (2010). A primer on metagenomics. *PLoS Comput Biol*, **6**(2), e1000667.

Worden, A. Z., Dupont, C., and Allen, A. E. (2011). Genomes of uncultured eukaryotes: sorting facs from fiction. *Genome biology*, **12**(6), 117.

Woyke, T., Xie, G., Copeland, A., Gonzalez, J. M., Han, C., Kiss, H., Saw, J. H., Senin, P., Yang, C., Chatterji, S., *et al.* (2009). Assembling the marine metagenome, one cell at a time. *PLoS One*, **4**(4), e5299.

Yang, X., Dorman, K. S., and Aluru, S. (2010). Reptile: representative tiling for short read error correction. *Bioinformatics*, **26**(20), 2526–2533.

Yang, X., Chockalingam, S. P., and Aluru, S. (2013). A survey of error-correction methods for next-generation sequencing. *Brief Bioinform*, **14**(1), 56–66.

Yoon, H. S., Price, D. C., Stepanauskas, R., Rajah, V. D., Sieracki, M. E., Wilson, W. H., Yang, E. C., Duffy, S., and Bhattacharya, D. (2011). Single-cell genomics reveals organismal interactions in uncultivated marine protists. *Science*, **332**(6030), 714–717.

Zerbino, D. R. and Birney, E. (2008a). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, **18**(5), 821–829.

Zerbino, D. R. and Birney, E. (2008b). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res*, **18**(5), 821–829.

Zong, C., Lu, S., Chapman, A. R., and Xie, X. S. (2012). Genome-wide detection of single-nucleotide and copy-number variations of a single human cell. *Science*, **338**(6114), 1622–1626.