Polymorphism and Genome Assembly

by

Nilgün Dönmez

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Polymorphism and Genome Assembly

Nilgün Dönmez

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2012

When Darwin introduced natural selection in 1859 as a key mechanism of evolution, little was known about the underlying cause of variation within a species. Today we know that this variation is caused by the acquired genomic differences between individuals. Polymorphism, defined as the existence of multiple alleles or forms at a genomic locus, is the technical term used for such genetic variations.

Polymorphism, along with reproduction and inheritance of genetic traits, is a necessary condition for natural selection and is crucial in understanding how species evolve and adapt. Many questions regarding polymorphism, such as why certain species are more polymorphic than others or how different organisms tend to favor some types of polymorphism among others, when solved, have the potential to shed light on important problems in human medicine and disease research.

Some of these studies require more diverse species and/or individuals to be sequenced. Of particular interest are species with the highest rates of polymorphisms. For instance, the sequencing of the sea squirt genome lead to exciting studies that would not be possible to conduct on species that possess lower levels of polymorphism. Such studies form the motivation of this thesis.

Sequencing of genomes is, nonetheless, subject to its own research. Recent advances in DNA sequencing technology enabled researchers to lead an unprecedented amount of sequencing projects. These improvements in cost and abundance of sequencing revived a

greater interest in advancing the algorithms and tools used for genome assembly. A majority of these tools, however, have no or little support for highly polymorphic genomes; which, we believe, require specialized methods.

In this thesis, we look at challenges imposed by polymorphism on genome assembly and develop methods for polymorphic genome assembly via an overview of current and past methods. Though we borrow fundamental ideas from the literature, we introduce several novel concepts that can be useful not only for assembly of highly polymorphic genomes but also genome assembly and analysis in general.

To dad,

- for always believing in me.

Acknowledgements

"So long, and thanks for all the fish."

- Douglas Adams, 1984

I am grateful for the help of many people who have shaped my life as a PhD student. First, I would like to thank my advisor Michael Brudno for his guidance and support. He has been a great mentor and role model for me through this entire time. His positive attitude often provided me with the motivation I needed in the face of challenges. Without his insights and extensive knowledge of the field, this thesis would not be possible.

In addition, I am greatly thankful to my past and present committee members: Ryan Lilien, Richard Zemel, Mark Braverman, Quaid Morris, Derek Corneil and Asher Cutter. Their insightful comments shaped the work presented in this thesis. Ryan Lilien, Richard Zemel and Mark Braverman guided me during my transition from MSc to PhD, while Quaid Morris, Derek Corneil and Asher Cutter helped build my research as a PhD student. From my many meetings with them, I learned how to ask and answer the right questions. I also would like to thank my external reviewer Daniel Brown for his many suggestions about the manuscript of this thesis and my collaborators Georgii Bazykin and Alexey Kondrashov for introducing me to an exciting research area, from which most of the work in this thesis originates.

I owe many thanks to my friends and fellow colleagues at the University of Toronto: Orion Buske, Matei David, Misko Dzamba, Marc Fiume, Justin Foong, Marta Girdae, Yue Jiang, Aziz Mezlini, Maria Mirza, Izhar Wallach, Joe Whitney, Vladimir Yanovsky, Recep Colak, Hilal Kazan, Sihui Asuka Guan and many others whose cheerful support have made this long journey easier. Finally, I am grateful to my family for the relentless love and support they have given me through the years. Their encouragement has been invaluable to me, both as a person and a student.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Launched in October 1990, the Human Genome Project (HGP) remains one of the largest collaborative projects in the history of science. Since the working draft of the human genome was announced in 2000, innovations in genome sequencing technologies along with the rapid increase in computational power due to Moore's Law suggest that sequencing of new species may be routine experiments in the next decade.

Due to technical limitations, sequencing a genome requires the combination of lab experiments with computational methods. *Genome assembly*, the name given to the computational part of sequencing, is the main subject of this thesis.

Genome assembly has been the topic of great interest since the first sequencing method was developed in 1975 [65], long before the HGP. While our work draws ideas from this plethora of research, our methods target a specific challenge in genome assembly: the presence of high polymorphism. *Polymorphism*, the existence of genetic variations within a population, is a recurring theme in this thesis.

In this chapter, we introduce polymorphism and genome assembly as two underlying subjects of this thesis and briefly explain how they relate to each other. In successive chapters, these subjects are handled in greater detail. An outline of this thesis and related publications are provided at the end of this chapter.

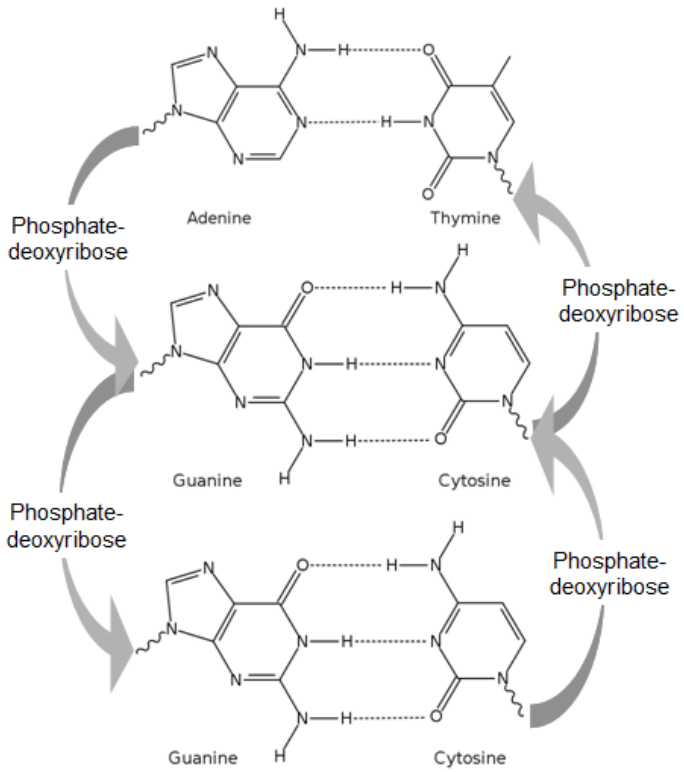

Figure 1.1: *Schematic view of a DNA molecule.* Each Adenine molecule makes two hydrogen bonds with a Thymine molecule, while each Guanine-Cytosine pair make three hydrogen bonds. Every nucleotide pair (also called a base pair) is connected to the next one by a phosphate-deoxyribo group on each side.

1.1 Background

1.1.1 Genomes and polymorphism

From a computational perspective, a DNA (Deoxyribonucleic Acid) molecule can be thought of as a word made of an alphabet of only four letters. These letters, in fact, represent four distinct unit molecules: Adenine, Cytosine, Thymine and Guanine. Each of these molecules is called a "nucleotide" or a "base" and is briefly denoted by its initial (i.e. A, C, T, G). When a number of these nucleotides form a sequence by making bonds, this larger molecule is referred to as a DNA molecule or sequence. In a DNA

molecule, nucleotides actually form two strands like the teeth of a zipper; where Adenine always couples with Thymine and Cytosine with Guanine (see Figure 1.1). Because of this coupling, DNA is typically measured in "base pairs". Since the coupling of the nucleotides is definitive, one strand of a DNA molecule completely determines the other strand. The sequences of the two strands are called the "reverse complements" of each other.

The *genome* of an organism is the name given to the DNA sequence contained in each somatic[1] cell of an organism. In prokaryotic organisms (e.g. bacteria) the genome is typically present as a single sequence, while in eukaryotes (e.g. animals, plants, fungi) the genome is divided into several chromosomes of varying sizes. In most eukaryotes, each somatic cell contains multiple copies of these chromosomes. For instance, most mammals have two copies of each chromosome, one inherited from each parent, while many plants have more than two copies. This copy number is referred to as *ploidy*. Organisms are often named by their ploidy as *diploids* (ploidy=2), *triploids* (ploidy=3), *tetraploids* (ploidy=4) and so on. An organism with a ploidy number of one is called *haploid* (e.g. bacteria, male bees and ants).

In the general sense, *polymorphism* refers to the existence of different "morphs" within a population of a species. When these morphs are observable (e.g. fur color in cats, blood type in human), they are called *phenotypes*. In this thesis, however, we focus on the genetic morphs between individuals, called *genotypes*, and use the term polymorphism to refer to these genotypic differences.

Genetic polymorphism in a population can exist in multiple forms. These can be grouped under two broad categories: nucleotide diversity and structural variation. Nucleotide diversity is largely based on nucleotide mutations called Single Nucleotide Polymorphisms (SNPs). These mutations emerge in gamete cells (i.e. egg or sperm cells) and are thus passed along to the offspring. SNPs contribute to the overall genetic diversity

[1]For multi-cellular organisms, a somatic cell usually means any cell other than the reproductive cells.

to a variable extent in different species. For instance, the nucleotide diversity (a.k.a the SNP rate) between humans is about 0.1%, while this rate can be as high as 3-5% in some fish and other sea organisms.

Structural variations typically refer to large scale events and are mainly based on sequence insertions, deletions and inversions. Like SNPs, these events have to emerge in gamete cells in order to contribute to the polymorphism in a population. The scale and rate of these events also vary across species yet their specifications are less well understood. Nonetheless, it is believed that structural variations contribute more to the overall genetic diversity than SNPs in complex organisms.

1.1.2 Sequencing

A substantial amount of research in biology today would not be possible without DNA sequencing; that is, the experimental determination of the sequence of a DNA molecule. Presently, no sequencing technology allows the identification of an entire genome. Sequencing is thus performed by first shearing a genome of interest into many fragments. These fragments are then subjected to various procedures - depending on the technique used, in order to determine their sequences. Each sequenced fragment is called a *read*.

Since its introduction in late 70s, sequencing has been dominated by the Sanger method [65]. Also called "capillary sequencing", the Sanger method starts by shearing many copies of a genome into small fragments using enzymes. Using a host organism such as bacteria, each fragment is cloned in large numbers. This procedure is called *amplification*. These cloned fragments are then read via a technique called primer extension using nucleotides labeled with fluorescent dyes. The task of determining the nucleotide sequence of a fragment by observing the light intensities of these dyes is called *base calling*. Due to limitations of the cameras used in this process and other complicating factors, base calling is not 100% accurate. Typically, Sanger reads contain low quality base calls at the beginnings and ends. Experimental anomalies, especially due to the amplification

Table 1.1: *The features of some available sequencing technologies at a glance.*

		Read length (bp)	Throughput (mbp/day)	Cost[c] ($/mbp)
	Sanger	800	6	500
2010[a]	Roche/454	400	750	20
	Illumina	100	5000	0.5
	SOLiD	50	5000	0.5
2008[b]	Roche/454	250	300	80
	Illumina	32	325	6
	SOLiD	35	600	6

[a] Estimates based on [37]. [b] Estimates based on [45]. [c] Note that the estimates for the year 2008 also includes amortized costs such as sequencing instrument acquisition and maintenance.

step, also contribute to errors in the reads. These errors can emerge as substitutions (i.e. miscalled bases) or indels (insertions/deletions).

The Sanger method allows sequencing of up to ∼400 reads in parallel, varying from around 600 to 1000 base pairs (bp) in length. Currently, commercial instruments using this technology can yield approximately 6 million base pairs (mbp) in one day, each million costing about $500. Given these cost and throughput figures, sequencing of large genomes have been expensive, long term projects and only performed by a small number of specialized sequencing centers.

Developed in the recent years, a number of new technologies have revolutionized DNA sequencing by increasing this throughput an order of magnitude higher while significantly reducing the cost at the same time. Various features of these sequencing platforms are summarized in Table 1.1.

The Roche/454 sequencing platform, commercially released in 2005, is based on a technology called "pyrosequencing" [60]. The distinctive feature of this technology is that, in each cycle, the four nucleotide types are read in order. For instance, in the first cycle the instrument might read **T:0, A:2, C:3, G:1** corresponding to **AACCCG**, in the second cycle it might read **T:3, A:1, C:0, G:1** corresponding to **TTTAG** and so on. This sequential nature of this platform ensures that substitution errors are rare. On the other hand, homopolymer (e.g. "AAAAA") lengths can not be identified with high accuracy and a large fraction of errors produced by this platform tend to be indels. The read lengths are determined by the number of cycles and the composition of the nucleotides in each fragment. Currently, Roche/454 instruments can perform around 200 flow cycles, producing an average read length of 400bp.

Following 454 Pyrosequencing, the Illumina Genome Analyzer was introduced as a sequencing platform based on the concept of "sequencing by synthesis"[6]. Unlike the technologies mentioned above, Illumina produces fixed length reads. Initially, this technology was only capable of producing very short sequences ($\sim$32) and had similar throughput to Roche/454, however, a number of technical improvements now allow this technology to produce >100bp long reads with higher throughput. In contrast with Roche/454, the majority of the errors produced by this platform are substitutions.

A third sequencing technology, commercially available from Applied Biosystems with the name SOLiD, is based on "sequencing-by-ligation"[68]. Unlike the other three technologies, this technology is based on reading overlapping "pairs" instead of single bases. Although there are 16 possible nucleotide pairs, only four different dyes are used to identify them. Due to this limitation, SOLiD instruments output a sequence of "colors" which can only be converted to nucleotide space using a known first letter. This unique "color space" sequencing of the SOLiD platform, despite having a relatively higher error rate, has some computational advantages when a reference sequence is available [61]. SOLiD instruments have similar throughput to Illumina, though the read lengths are still limited

to ∼50bp.

All current sequencers produce a quality score with each base. This "Phred" score, named after the base calling tool [22] developed for Sanger sequencing in 1998, is based on the estimated probability of the corresponding base being misidentified. Formally, the relation between the Phred score and this probability of error is given by $p = 10^{\frac{-q}{10}}$, where p is the probability and q is the Phred score. By convention, Phred scores are always discrete numbers. While the range of Phred scores varies slightly across platforms, they usually fall between 0-60. Quality scores are useful in a variety of applications from SNP detection to genome assembly and exploited by a number of tools (e.g. [35, 20]).

Apart from the aforementioned technologies, a few other sequencing platforms, such as HeliScope by Helicos and SMRT by PacBio, have also been made available in the past couple of years. These technologies target Single Molecule Sequencing (SMS), which allows sequencing of DNA molecules without an amplification step; however, they have significantly higher error rates. In this thesis, we focus on the general sequencing platforms as described above. For an extensive review of these and other sequencing platforms and their applications, we refer the reader to [45, 57, 37].

Paired sequencing

The new sequencing platforms are rapidly replacing the conventional Sanger sequencing due to their dramatically reduced cost and high throughput. As Table 1.1 demonstrates, these technologies are still developing. For example, between the years 2008-2010, the read lengths have nearly doubled, the throughput increased several fold and the cost dropped further. On the other hand, the read lengths are still significantly shorter than Sanger sequencing, which complicates certain applications.

Paired sequencing protocols may help overcome some of the limitations of these short reads. Paired sequencing involves determining the sequence of reads in pairs, where the approximate separation between the reads is known. Paired reads can be generated using

either of the paired-end or mate-pair protocols.

The paired-end protocol is primarily performed in the Illumina platform. This protocol involves reading both ends of a fragment that is slightly larger than the read size. Since this protocol is fairly easy to perform, almost all Illumina sequencing consist of paired-end reads. On the other hand, the size of the fragments are limited to <1000bp.

The mate-pair protocol allows fragments to be much larger (e.g. 8, 12 or 20 kb), yet, it requires the preparation of specific libraries and is more tedious. Furthermore, depending on the technique, a fraction of the reads produced by this protocol may be "chimeric"; meaning that the read is composed of two separate sequences mistakenly concatenated. In principle, the mate-pair protocol can be applied to any of the four sequencing technologies described above, although it is less frequently applied in the new platforms due to the tedious library preparation stage.

In neither of these protocols, can the size of the fragments be determined with high precision. In particular, this precision depends on the insert size (i.e. the intended fragment size). As a result, the standard deviation is often stated as a percentage of the mean insert size and typically lies in the range 10-20%.

1.1.3 Genome assembly

As discussed in the previous section, available sequencing technologies can read DNA fragments of no more than several hundred base pairs. In contrast, most living organisms have genomes orders of magnitude longer, ranging from a few million base pairs in bacteria to billions in plants and mammals.

Determining the whole sequence of a genome requires combining these experimental technologies with computational methods. As we have mentioned previously, sequencing a genome involves shearing the genome into many small fragments. Though certain biases exist, this shotgun sequencing process - as it is often called - can be viewed as a random sampling of the genome. To ensure every base of the genome is covered by at

least one read with high probability, a level of over-sampling is necessary. The extent of this over-sampling is called "coverage" and defined as the ratio of the total number of the base pairs in reads to the length of the genome. For example, 60 million 100bp-length reads sampled from a human genome (3 billion bp) implies 2× coverage.

The assembly problem is the *in silico* reconstruction of the whole genome given the reads sampled as described above. The difficulty of genome assembly lies in several factors. Arguably, the most important factor is the ratio of the size of the DNA fragments, i.e. the length of the reads, to the length of the genome. Similar to a jigsaw puzzle, smaller pieces imply a larger number of them is needed to construct a solution thus increasing the complexity of the problem. In addition, this increase in complexity is non-linear; that is, assembling a genome with reads of length 50bp is often more than twice as hard as assembling the same genome with 100bp reads. This non-linearity is primarily due to genomic repeats - stretches of DNA that appear in identical or near-identical form at two or more locations in the genome. Repeats that are longer than reads pose significant problems for genome assemblers. Yet, most eukaryotic genomes have large (>1kbp) genomic repeats.

A second factor that contributes to the complexity of genome assembly is sequencing inaccuracy. As we note in the previous section, all sequencing technologies produce a certain fraction of errors. This inaccuracy may be mitigated via additional coverage. On the other hand, increasing coverage not only adds to the cost and time of sequencing but also creates technical challenges due to the increased volume of sequencing data that need to be stored and processed in disk/memory. Consequently, many genome assemblers have specialized methods to deal with sequencing errors [9, 79, 20]. Several stand-alone error-correction tools are also available for use either as a precursor to an assembler or simply to improve sequencing data accuracy prior to downstream analyses (e.g. [62, 29, 35, 64, 32]).

A less recognized, yet fundamental, factor that affects genome assembly is polymor-

phism. Though genomic differences between species do not play a role in sequencing and assembly, with the exception of metagenomics [41], a high level of within-population polymorphism may significantly increase the complexity of genome assembly.

For diploid or polyploid organisms, the population level of polymorphism is reflected within each individual's genome as each cell contains two or more copies of the chromosomes. In a general context, it is not possible to sequence these chromosomes separately and the sequencing data includes reads from all copies. If the differences between these copies involve only a small fraction of the bases, they have little impact on the assembly process. Indeed, for most organisms sequenced to date, polymorphism has been ignored and a single reference sequence is produced, which is usually a mosaic of the parental chromosomes. Different types of polymorphism are then detected via sequencing of additional individuals or other experimental techniques and added to specialized databases such dbSNP (`http://www.ncbi.nlm.nih.gov/projects/SNP/`).

If the parental chromosomes differ in a large fraction of bases, genome assembly is negatively affected since the differences can no longer be ignored. Several sequencing projects in the past few years revealed a number of organisms with polymorphism levels higher than previously observed [14, 74, 72]. The assembly of these genomes using off-the-shelf tools proved difficult, requiring extensive manual intervention and modification of existing algorithms [76, 73].

As increasing read lengths is solely in the realm of experimental improvements, the methods we present in this thesis target the latter two challenges; namely, sequencing errors and polymorphism. In particular, we will see that these challenges interact with each other in a way that each challenge makes the solution of the other problem harder.

Another factor that may significantly impact genome assembly is paired sequencing. Unlike the challenges described above, paired reads, when available, have a positive impact on genome assembly. When the separation between the pairs is long enough, paired reads have the potential to resolve more repeats, thus creating better assemblies.

Another use of paired reads is "jumping" across coverage gaps. This task is called *scaffolding* and is performed as a post processing step. Scaffolding allows longer sequences to be assembled. In this thesis, we present novel methods for both uses of paired sequencing.

1.2 Outline

In the next chapter, we review the existing sequencing error correction methods and present a novel method based on Naive-Bayes that can handle high polymorphism rates. We evaluate our method, named ENCORE, using both simulated and real reads in different platforms. In Chapter 3, we review common approaches to genome assembly and introduce a novel structure called the "Mate Pair Graph" that is designed to exploit paired read information in order to resolve repeats and polymorphism. We evaluate Hapsembler, our genome assembler that implements this approach, on a highly polymorphic sea ascidian named *Ciona savignyi*. We also compare Hapsembler to existing genome assemblers on a haploid genome.

We complete our discussion of genome assembly in Chapter 4, where we summarize the available scaffolding methods and present novel algorithms to perform this task. We compare these algorithms, implemented as a stand-alone scaffolder named ScaRPA, to several other scaffolders on two bacterial genomes. In addition, we evaluate the combined performance of Hapsembler and ScaRPA against another genome assembler that has support for scaffolding.

In Chapter 5, we showcase a study of polymorphism in *C. savignyi*. This case study illustrates the diversity of research applications that can be performed on such highly polymorphic genomes, providing a motivation for the methods presented in the previous chapters. Finally, Chapter 6 concludes this thesis with a brief note.

1.2.1 Relations to other publications

Parts of this thesis have been previously published. Parts of Chapter 2 and Chapter 3 are published in [20] (see Copyright Notice below for details). The methods and results of Chapter 5 including Tables 5.1, 5.2 and 5.3 are published in [19]. Chapter 4 is currently under preparation as a journal manuscript.

All work in this thesis is done in collaboration with Michael Brudno. In addition, Chapter 5 is joint work with Georgii Bazykin and Alexey S. Kondrashov.

Copyright Notice:

Chapter 2

Error correction in the presence of high polymorphism

2.1 Introduction

Although the fractions vary, all sequencing technologies produce erroneous reads. In genome assembly, these erroneous reads present a dilemma: while error-free reads could help distinguish inexact repeats from each other, if we treat all disagreements between the reads as genuine differences, reads containing sequencing errors essentially become useless. In Chapter 3 we will see that some assemblers take the approach of ignoring such reads (or at least their error-prone portions), whereas others allow a small fraction of differences between the reads and resolve errors at a later stage. In either scenario, the positive effect of error correction on genome assembly has been demonstrated by numerous studies [56, 35, 64, 32].

In addition to assembly, sequencing errors complicate any application where a set of reads are mapped to a reference sequence. Due to efficiency issues, most mapping algorithms allow a limited number of disagreements between the reference and a read. As a result, reads containing multiple errors may not be mapped to a location. Further-

more, when a mapping location can be identified, it is very difficult to determine whether the disagreements between the read and the reference are due to errors or genuine nucleotide differences. For example, sequencing errors pose significant challenges in Single Nucleotide Polymorphism (SNP) detection[35].

Curiously, this is a dual problem: while polymorphism detection is hindered by the presence of sequencing errors, error correction itself is negatively affected from a high level of polymorphism. Most correction methods rely on uniform coverage and detect errors based on their low frequency. In the presence of polymorphism, this assumption leads to an excess of false negatives: if one allele happens to have low coverage, it may be mistaken for an error.

In this chapter, we momentarily shift our attention from assembly to sequencing error correction as a precursor to successful polymorphic genome assembly as well as other applications. In particular, we present a novel method to remove sequencing errors from reads in the presence of high polymorphism. This method is based on a simple but fundamental notion: if two otherwise similar reads disagree on a base and both base pairs have high quality scores, the disagreement is more likely due to a SNP rather than a sequencing error. To incorporate this idea in a robust probabilistic framework, we apply a well known machine learning method called Naive-Bayes [18].

The outline of this chapter is as follows. First, we give a brief summary of error correction tools currently available. In the following section, we elaborate on the details of our error correction approach. We then give a presentation of the results achieved by our method on various sequencing platforms and conclude the chapter with a discussion.

2.1.1 Related work

Currently, a plethora of stand-alone sequencing error correction tools exists that are applicable to specific platforms and error types. Shrec [67], which is based on a suffix trie built from reads, identifies low weight branches in the trie which are subsequently

altered to map high weight branches but can only handle substitution errors. A modified version of Shrec [62] is adapted to handle insertion and deletion errors as well as color space reads generated by the SOLiD platform. Hitec [29] uses a suffix array instead of a suffix trie to save space and perform a statistical analysis on the suffix array to correct errors.

Some assemblers such as Euler-SR [9] and Allpaths [8] incorporate read correction directly into their framework using a method called *Spectral Alignment* (SA) or *k-spectrum*. This method is based on the idea that low copy k-mers (i.e. length k subsequences of reads) are likely to be erroneous and can be corrected by making a small number of edits so that they are converted into high copy k-mers. Although SA works well with de Bruijn graph based assemblers, it is less accurate and ill-fitted for general purpose read correction. This is due to the fact that this method is quite sensitive to non-uniform coverage and prone to corrupting genuine k-mers.

Criticizing the indifference of the k-mer frequency approaches to base qualities, Kelley *et al.* [35] has introduced a method that also utilizes the quality scores. This stand-alone error correction algorithm, named Quake, leverages the k-mer coverage framework by prioritizing the alteration of bases with low quality scores.

Most of the aforementioned methods rely on a uniformly high coverage to correct sequencing errors. An exception is Hammer [49], which is developed for applications such as single-cell or transcriptome sequencing, where coverage is drastically non-uniform [39, 11, 42]. Hammer is based on the idea of building a Hamming graph of k-mers and analyzing the clusters in this graph. Nonetheless, this method is still susceptible to making mistakes if the genome contains a large number of near-exact repeats or is highly polymorphic.

Recently, Salmela and Schröder [64] have developed a multiple sequence alignment based framework named Coral, that is applicable to virtually any sequencing platform. However, their method performs a simple majority voting to correct errors and is still

prone to overcorrecting SNPs and small genuine indels.

In summary, despite the fact that a variety of error correction approaches exist, polymorphism is usually ignored and the methods are only tested on bacterial or other haploid genomes. Even when diploid organisms are represented [32], the polymorphism rates considered are under 1-2%, which falls below the levels targeted in this thesis.

In the rest of this chapter, we present a method to effectively remove sequencing errors when a large fraction of bases have multiple alleles. This method, which we shall refer to as ENCORE (Effective Naive-Bayesian Correction of Errors), is similar to Coral in that both methods form multiple sequence alignments to detect and correct errors. On the other hand, there are several differences between the two algorithms in how they perform sequence alignment and treat quality scores. As we show below, these differences make ENCORE more resistant to corrupting genuine bases. Another framework similar to ENCORE has recently been implemented to correct reads in PANDAseq [46]. However, in PANDAseq the error correction is only performed between the two ends of a paired-end fragment and therefore polymorphism is not an issue.

ENCORE has been implemented in C++ as a part of the genome assembly toolkit Hapsembler [20], though it can be used as a stand-alone program to correct reads for any purpose (see `http://compbio.cs.toronto.edu/hapsembler` for details).

2.1.2 Motivation for error correction via sequence alignment

We have previously noted that SNPs and short indels complicate the error correction process because they masquerade as sequencing errors. Here we show how sequence alignment may alleviate the effect of SNPs and genuine indels with the help of quality scores. Consider the pairwise sequence alignment of two reads shown in Figure 2.1. In the first two mismatches, both reads have high quality scores suggesting that these may in fact be SNPs. In the third mismatch, one of the reads has a low score, which may or may not be due to a sequencing error.

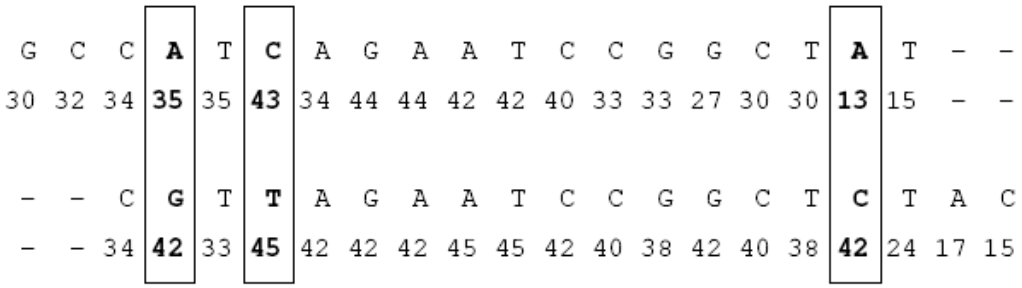

Figure 2.1: *A pairwise alignment of two reads with three mismatches.* The base qualities associated with the first two mismatches are high suggesting that these might be genuine alleles. On the other hand, in the third mismatch, one of the bases has a low score, which may or may not be due to a sequencing error.

In a k-mer coverage framework, information about far apart mismatches between two reads does not propagate. In the case shown in Figure 2.1, no k-mer will span the last two columns if the k-mer size is 11 or less. Therefore, any algorithm based on k-mer coverage will make decisions for these columns separately. If sequence coverage is perfectly uniform and high, this may not pose a problem since SNPs will in general have more support than sequencing errors. Nevertheless, even in relatively high coverage sequencing projects, coverage is variable and low coverage regions are not uncommon. Consequently, low coverage regions are either corrupted by these methods or not corrected at all.

In contrast, a sequence alignment based method can incorporate full information about the reads without having to rely on high coverage when SNPs and short indels are prevalent. In particular, when correcting a query read, we can weigh the contribution of each read based on the quality of the overall pairwise alignment between the query and the latter. This notion lends itself to an elegant Naive-Bayes framework as we explain in the next section.

2.2 ENCORE algorithm

2.2.1 Overlap computation

To perform correction, we first need to detect overlaps (i.e. alignments) between the reads. Considering every pairing in n reads - requiring $O(n^2)$ comparisons - is prohibitive for most datasets. A common technique to accomplish this task efficiently is k-mer indexing. In its basic form, this technique involves building a hash of k-mers sampled from the reads starting at every base, and then comparing only those reads sharing one or more k-mers. Each hash entry includes the list of reads containing the k-mer, and in most implementations also the position of the k-mer within the read.

For large datasets, storing this index is costly. To reduce the space requirement, ENCORE employs a variation of this technique inspired by the work of Rasmussen *et al.* [58]. In this variation, we build the hash by sampling k-mers at every k^{th} position of a read. As Rasmussen *et al.* show, this reduced sampling is sufficient to find all ϵ-matches between the reads provided that k is small, where ϵ is twice the expected error rate [58]. This is achieved by calculating the number of shared k-mers between two reads required to ensure that the reads have an alignment with identity $(1 - \epsilon)$ or more over a given length. When the k-mers are sampled at every k^{th} interval, this calculation is trivial. For example, if $k = 13$ and $\epsilon = 0.02$, two reads should share at least three k-mers in order to have an overlap of length 52 or more, regardless of where the errors occur.

To correct a read, we consider **all** k-mers contained in the read and look up these k-mers in the index. We create a list of all reads encountered in this process together with the number of shared k-mers and their relative positions in both reads. Storing the locations of the shared k-mers enables us to estimate the expected length of the overlap between the two reads. We then use the formula $\frac{l_{ovl}}{k} - (l_{ovl} \times \epsilon)$ to compute the minimum number of k-mers required to be shared between the read and the query; where l_{ovl} is the estimated length of the overlap. In general, distinct shared k-mers may suggest different

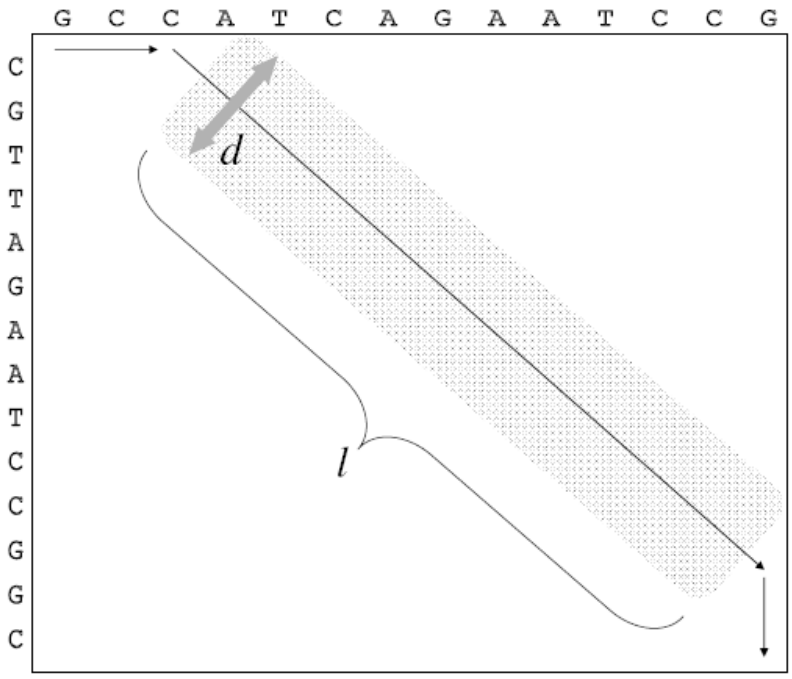

Figure 2.2: *Modified Needleman-Wunsch algorithm.* If we expect the best alignment to occur along the diagonal arrow, we only compute the cells within a distance $d = l_{ovl}t$ from this diagonal, where l_{ovl} is expected the overlap length and t is the error tolerance.

overlap lengths. If the inferred overlap lengths and locations are close, we bundle these k-mers and take the average, otherwise they are considered separately. To avoid overlaps that occur by chance, we impose a minimum overlap length l_{min}.

Once we decide which reads to compare with the query, we use a modified version of the Needleman-Wunsch (NW) algorithm [54] to align these reads against the query. Originally, this algorithm takes two sequences of lengths l_i and l_j and returns the best global alignment between them using a dynamic programming matrix of size $l_i \times l_j$. Thus, the running time of the original algorithm is quadratic in read lengths.

To speed up this task, we modify the algorithm so that only the relevant parts of the dynamic programming matrix are calculated. Again, we use the expected length and location of the overlap inferred from the shared k-mers. The matching k-mers suggest that the best alignment should occur along a certain diagonal in the NW matrix. Thus, we

only compute those cells within a distance d from this diagonal. d denotes the maximum number of disagreements we are willing to allow in the overlap and is calculated as $l_{ovl}t$, where l_{ovl} is the expected overlap length and t is the tolerance rate for errors (Figure 2.2). Since base mismatches do not require a deviation from the diagonal, if indel type errors are rare, t may be set to a lower value than ϵ. This yields significant savings in computation time for sequencing platforms such as Illumina, where the dominant type of error is substitution.

If the modified NW algorithm returns an overlap with length $\geq l_{min}$ and at least $(1-\epsilon)$ identity, we store the pairwise alignment for further processing. Otherwise, we discard the alignment. The entire process is repeated once more for the reverse complements of the k-mers found in the query read. This time, we take the reverse complements of the matching reads before performing sequence alignment.

While the k-mer hash can be built in time linear in m, the total number of bases, the running time of the overlap computation stage is dominated by the NW calls. In the worst case, we may have to compare all pairs of reads leading to a time complexity of $O\left(l^{1+t}n^2\right)$, where n is the number of reads, l is the read length and t is the error tolerance rate. In practice, this worst case scenario is almost never seen. Nevertheless, if the genome is repeat-rich, reads from high-copy repeats significantly increase the running time. Note that, the highest copy repeats are often short. These result in high-frequency k-mers, yet reads sharing such k-mers do not necessarily have a sufficient overlap. In order to avoid spending too much time on such reads, we impose a limit on the k-mer frequencies. If a k-mer reaches a frequency of f_{max}, we stop adding new reads to its hash entry. Thus, the maximum number of overlaps that can be detected for any read is bounded by $O\left(\frac{l}{k}f_{max}\right)$. Consequently, the running time of the overlapping stage can be stated as $O\left(\frac{l^{2+t}}{k}f_{max}n\right)$. We note that this bound is still far from being tight; as for the majority of the reads the number of overlaps remains directly proportional to the sequence coverage.

Setting the parameters for overlap computation

Since overlap computation is the bottleneck of our error correction algorithm, it is important to state the parameters that control the trade-off between its sensitivity and speed. These parameters are k, the k-mer size, l_{min}, the minimum overlap length and f_{max}, the maximum k-mer frequency.

Recall that we require at least $\frac{l_{ovl}}{k} - (l_{ovl} \times \epsilon)$ shared k-mers between two reads in order to perform sequence alignment. If ϵ is high and k is set too large, this formula may return a value less than 1. Naturally, it is not possible to detect an overlap if no k-mer is shared. In contrast, if it is set too small, we make many NW calls that do not yield valid alignments. A rule of thumb therefore is to set k to the maximum value that satisfies $\frac{l_{min}}{k} - (l_{min} \times \epsilon) \geq 1$.

In turn, the minimum overlap length, l_{min}, should be set based on the sequence coverage and the average read length. When l_{min} is too large, fewer overlaps are detected reducing the effectiveness of error correction. When it is too small, the running time increases since we have to perform more sequence alignments. A reasonable range for this parameter can be derived statistically. If we assume all reads have length l and are randomly sampled from the genome, then the expected number of reads with no overlapping neighbours can be estimated as $ne^{-2c(1-l_{min}/l)}$, where c is the sequence coverage [50]. Ideally, we would like this number to be as small as possible, hence l_{min} should be set accordingly.

Although it is difficult to derive an optimal value for f_{max}, in practice, we have found that its effect is minimal for most datasets so long as it is set to a reasonably high value (e.g. $\sim$5k) and increased linearly with coverage.

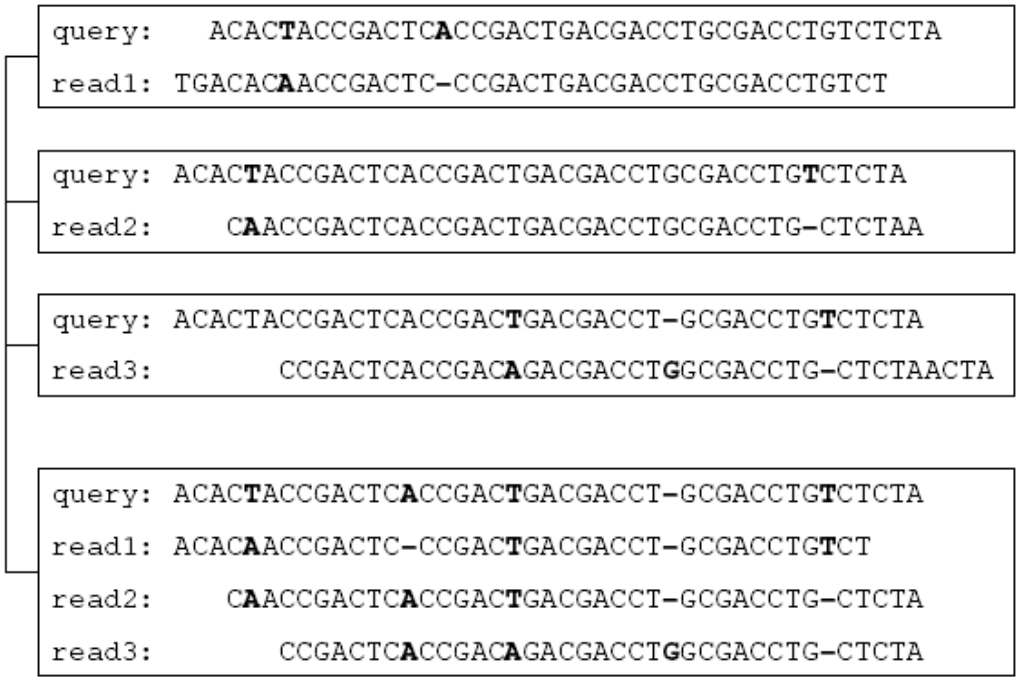

Figure 2.3: *An example Multiple Sequence Alignment (MSA) using the star heuristic.* The pairwise alignments between the query and the other reads are combined to generate a pairwise-consistent MSA. The columns that contain disagreements are typed in boldface for visualization. Note that only the parts of the reads that align to the query are used in generating the MSA.

2.2.2 Multiple sequence alignment

The overlap computation stage generates a set of pairwise alignments for the read to be corrected. In the next stage, we construct a multiple sequence alignment (MSA) of these reads. As finding an optimal MSA is computationally impractical, we use a heuristic algorithm called "star alignment" [2]. In essence, this algorithm constructs an MSA around a reference read that is consistent with all the pairwise alignments of the reference to other reads (see Figure 2.3). This heuristic is suitable for our purpose since we already have a natural choice of reference, i.e. the read to be corrected, and its pairwise alignments to other reads. In addition, if all the reads are indeed from the same location in the genome, we expect little deviation among them justifying the use of a simple heuristic algorithm.

2.2.3 The Naive-Bayes probabilistic model

The Naive-Bayes model is a simple probabilistic model based on the Bayes' Theorem with a "naive" independence assumption. Let C be a random variable that is conditional on a feature set $\{F_1, F_2, ..., F_n\}$. The Bayes' Theorem states that:

$$p(C|F_1, F_2, ..., F_n) = \frac{p(C)p(F_1, F_2, ..., F_n|C)}{p(F_1, F_2, ..., F_n)} \quad (2.1)$$

$$= \frac{p(C)p(F_1|C)p(F_2|C, F_1)...p(F_n|C, F_1, F_2, ..., F_n)}{p(F_1, F_2, ..., F_n)} \quad (2.2)$$

The formula above becomes intractable as the number of features grows. If we assume that each feature F_i is conditionally independent of every other feature given C, the right hand side of the equation simplifies to:

$$p(C|F_1, F_2, ..., F_n) = \frac{p(C)p(F_1|C)p(F_2|C)...p(F_n|C)}{p(F_1, F_2, ..., F_n)} \quad (2.3)$$

$$= \frac{1}{Z}p(C)\prod_{i=1}^{n} p(F_i|C) \quad (2.4)$$

This assumption of conditional independence forms the basis of all Naive-Bayes models. Above, $Z = p(F_1, F_2, ..., F_n)$ is called the normalization factor, which ensures that we have a valid probability distribution function, and is typically ignored when the Naive-Bayes model is used as a classifier.

2.2.4 Naive-Bayesian error correction

In our error correction algorithm, we adapt the Naive-Bayes model at two levels. At the lower level, we use this model to estimate the probability that two reads are sampled from the same location of the same haplotype (i.e. haploid genotype), given their pairwise alignment. We refer to reads that belong to the same haplotype and location as "neighbours". At the higher level, we use this model to produce a consensus sequence for the query read, given the multiple sequence alignment and the likelihood of each read being a true neighbour of the query. For ease of narration, we start with the higher level.

Formally, let $C_x \in \{A, C, G, T\}$ denote the x^{th} base of a read R. Suppose that R has pairwise alignments with the reads $\{S^1, S^2, ..., S^n\}$. Let F_i denote the base aligned with the x^{th} base in the pairwise alignment between R and S^i. Then the conditional probability of C_x assuming a particular nucleotide is given by:

$$p(C_x|F_{i=1,2,...,n}) = \frac{1}{Z} p(C_x) \prod_{i=1}^{n} p(F_i|C_x) \tag{2.5}$$

Above, $p(C_x)$ is the prior probability of C_x and it is equal to $1 - 10^{-q/10}$ if it has the same value as the nucleotide present in the read, where q is the Phred score [22] associated with the base. Otherwise it is equal to $(10^{-q/10})/3$. $p(F_i|C_x)$ is given by the equation:

$$p(F_i|C_x) = p(F_i|C_x, B_i)p(B_i) + p(F_i|C_x, \neg B_i)p(\neg B_i) \tag{2.6}$$

$$= p(F_i|C_x, B_i)p(B_i) + p(F_i)p(\neg B_i) \tag{2.7}$$

where B_i is a binary variable indicating whether the two reads are neighbours or not. When two reads belong to different loci, we assume that their bases are independent of each other hence $p(F_i|C_x, \neg B_i) = p(F_i)$ (i.e. we have conditional independence). $p(F_i|C_x, B_i)$ equals $1 - 10^{-q_i/10}$ if $F_i = C_x$ and $(10^{-q_i/10})/3$ otherwise, where q_i is the quality score associated with F_i. $p(F_i)$ simply denotes the probability of seeing that particular nucleotide at that position and is given by $1 - 10^{-q_i/10}$.

In the equations above, we abuse the notation slightly and use $p(B_i)$ to denote the posterior probability of S^i and R being neighbours. To estimate $p(B_i)$, we first have to compute the following probabilities:

$$p(B_{R,S^i}|R, S^i) = \frac{p(B)}{Z'} \prod_{j=1}^{k} p(R_j, S^i_j|B) \tag{2.8}$$

$$\begin{aligned} p(\neg B_{R,S^i}|R, S^i) &= \frac{p(\neg B)}{Z'} \prod_{j=1}^{k} p(R_j, S^i_j|\neg B) && (2.9) \\ &= \frac{p(\neg B)}{Z'} \prod_{j=1}^{k} p(R_j)p(S^i_j) && (2.10) \end{aligned}$$

Above, k is the number of bases in the pairwise alignment between R and S^i. $p(R_j, S^i_j|B)$ denotes the conditional probability of the j^{th} position in the alignment. If two reads are indeed neighbours, the disagreements between their sequences should be due to sequencing errors alone. In other words, if two bases differ, at least one of them must be an error. Let q denote the quality score of the j^{th} base in read R and q^i denote the quality score of the corresponding base in read S^i. If $R_j \neq S^i_j$:

$$\begin{aligned} p(R_j, S^i_j|B) &= (1 - 10^{-q/10})((10^{-q^i/10})/3) && (2.11) \\ &\quad +(1 - 10^{-q^i/10})((10^{-q/10})/3) && (2.12) \\ &\quad +2((10^{-q/10})/3)((10^{-q^i/10})/3) && (2.13) \end{aligned}$$

If $R_j = S_j^i$, on the other hand, either both reads are correct or they both have a sequencing error:

$$p(R_j, S_j^i|B) = (1 - 10^{-q/10})(1 - 10^{-q^i/10}) \quad (2.14)$$

$$+(10^{-q/10})((10^{-q^i/10})/3) \quad (2.15)$$

$p(R_j)$ and $p(S_j^i)$ are computed as $(1 - 10^{-q/10})$ and $(1 - 10^{-q^i/10})$ respectively.

The prior probability $p(B)$, of two reads belonging to the same location is set to a value near 1.0 since we only compare reads that have a sufficient overlap with high identity. The posterior probability $p(B_i)$ is then estimated using the logistic function:

$$p(B_i) = \frac{1}{1 + \exp(\log p(\neg B_{R,S^i}|R, S^i) - \log p(B_{R,S^i}|R, S^i))} \quad (2.16)$$

The consensus nucleotide of read R for position x is chosen to be the nucleotide that gives the highest probability $p(C_x|F_{i=1,2,\ldots,n})$.

Above, we can ignore the calculation of the normalization factors Z and Z'. For the latter, the calculation is unnecessary since the two instances of Z' cancel each other in Equation 2.16. For the former, the calculation is unnecessary since we are only interested in the ranking of the probabilities and not their nominal values.

Note that the equations above do not account for indel errors. Although indels could be handled similarly, there are no associated quality scores with missing bases. We treat indels separately: If a significant fraction of reads are calling for a deletion the base is deleted. A similar rule is applied for insertion. If there is sufficient support for an insertion, the insertion base is selected using the same procedure as above using only those reads supporting the insertion. In this case, $\log p(C_x)$ is taken to be $\log(1/4)$, assuming an equal prior on each of the four nucleotides. For the computation of $p(B_i)$, we use a default gap quality score which depends on the sequencing platform.

2.2.5 Complexity analysis

As we have discussed in Section 2.2.1, the overall complexity of the overlap computation stage is $O\left(\frac{l^{2+t}}{k} f_{max} n\right)$. For each read, the number of columns in the MSA is bounded by $O\left(l + f_{max} l t\right)$. The latter term is due to the fact that each pairwise alignment can introduce at most lt gaps into the MSA. Thus, the overall space and running time required to compute the MSA is $O\left(f_{max} l (1 + f_{max} t)\right)$. If we assume all mathematical operations take constant time (i.e. $O(1)$), the time complexity of the Naive-Bayes computations is linear in the size of this MSA. Thus, the cumulative time complexity of the MSA and Naive-Bayes stages is $O\left(f_{max} l (1 + f_{max} t) n\right)$.

In practice, we employ several techniques to speed up the mathematical computations during the Naive-Bayes stage. First, the product operations are computed in log space. Furthermore, we create a look-up table to store frequently needed calculations. This is possible due to the discrete nature of the quality scores, which typically range between $0 - 60$. For example, we compute the Equations 2.13 and 2.15 for all quality scores in advance.

The space complexity of the methods is typically dominated by the k-mer index, requiring $O(m)$ space, where m is the total number of bases.

Note that once the k-mer index is created, the MSA and Naive-Bayes tasks are performed separately for each read, allowing them to be computed in parallel. In ENCORE, we parallelize these tasks for multi-core machines, achieving several fold speed up depending on the number of the cores.

2.3 Evaluation

In this section, we evaluate the performance of ENCORE along with a comparison to other error correction tools. Our tests span a variety of sequencing platforms and genomes with simulated and real reads. In particular, we assess the effect of sequence coverage

on error correction and show that at low coverage levels the careful treatment of polymorphism is critical. Our experiments on a bacterial genome also demonstrate that this seemingly conservative approach need not result in reduced effectiveness in correction when polymorphism is absent.

2.3.1 Datasets

We evaluate our methods on two organisms: *E. coli* and *C. savignyi*. *E. coli* is a well studied bacterium with a genome length of ∼4.6 million base pairs (mbp). We use a high quality reference sequence for *E. coli* which is available at NCBI (`http://www.ncbi.nlm.nih.gov/`) with the accession code NC_000913.2. *C. savignyi* is a sea ascidian and has a highly polymorphic diploid genome. Since a finished genome assembly is not available for this organism, we use a small portion of the draft assembly generated by Small *et al* [73] as our reference. This draft assembly is organized in 374 "hypercontigs", where each hypercontig is a pairwise alignment of two sequences, each representing a single haplotype. To use in our experiments, we choose the three largest hypercontigs, with a total size of 33mbp (haploid size = 16.5mbp).

2.3.2 Effect of sequence coverage on the accuracy of error correction

To test the effect of coverage on error correction, we simulate Sanger style reads from the reference sequence we constructed for *C. savignyi*. In order to make these simulations realistic, we use the Sanger reads from the original *C. savignyi* sequencing project `http://www.broadinstitute.org/annotation/ciona/` as templates. Simulation is performed by mapping the quality scores of each template to a random location of the reference and extracting that region as a read. This is repeated until the desired coverage depth is reached. For each base, we use the corresponding quality score to decide

Table 2.1: *Effect of coverage depth on error correction.*

Sequence Coverage	Total bases (mbp)	Err. after trimming (kbp)	Method	Err. after correction (kbp)	Mis-corrections (kbp)	Reduction (%)
7x	102	2,199	H-Shrec	2,893	1,453	-31.5
			ENCORE	741	51	66.3
10x	148	3,056	H-Shrec	4,370	2,443	-41.9
			ENCORE	631	48	79.3
13x	194	3,924	H-Shrec	4,750	2,221	-21.0
			ENCORE	598	45	84.7

whether to simulate a sequencing error. In accord with how Phred scores are interpreted [22], if the quality score of the base is q, we introduce an error with probability $10^{-q/10}$. Errors consist of substitutions, insertions and deletions with a distribution of 75%, 12.5% and 12.5% respectively. We simulate three coverage depths: 7x, 10x and 13x (Table 2.1).

Following the convention for Sanger sequencing, we trim the very low quality bases at the start and end of the reads before correction. Note that this is typically not sufficient to remove all errors. For instance, in our experiments, the error rate is around 2% after trimming (Table 2.1).

On these datasets, we compare the performance of ENCORE to the modified version of Shrec [62] (referred to as H-Shrec throughout this text). We choose this implementation of Shrec for its ability to handle indel errors. The results on three coverage levels are summarized in table 2.1. Observing this table, we see that H-Shrec introduces more errors than it corrects, most likely due to the high prevalence of SNPs and indels in this dataset. Interestingly, coverage depth seems to have a non-linear effect on its performance: The error reduction rate first decreases, then increases as we go from 7x to 13x sequence

Table 2.2: *The effect of error correction on real Roche/454 and Sanger reads.* Number of reads mapped is calculated by counting the reads that map with at least 95% identity and 95% coverage. A read is considered to be perfect if the entire read maps with 100% identity. The numbers in parenthesis denote the number of discarded reads (by H-Shrec) that map in each category. H-Shrec discards a total of 13,600 and 1,132 reads from the *C. savignyi* and *E. coli* datasets respectively.

Data	Error correction	No. of reads mapped	No. of reads mapped perfectly	Total size of perfect reads (mbp)
E. coli	None	88,624	4,154	1.6
	H-Shrec	(738) 88,814	(10) 9,573	3.7
	ENCORE	89,817	10,292	3.9
C. savignyi	None	411,626	20,689	13.6
	H-Shrec	(11,401) 391,016	(761) 48,994	32.6
	ENCORE	421,819	126,306	83.9

coverage. In contrast, ENCORE introduces very few errors and is able to reduce the number of errors by $66 \sim 85\%$. These results suggest that ENCORE is suitable for low coverage sequencing projects even in the presence of high polymorphism.

2.3.3 Assessment of error correction on different sequencing platforms

In the previous section, we have analyzed the effect of sequence coverage using simulated reads. In this section, we assess the performance of ENCORE on different sequencing platforms using real reads.

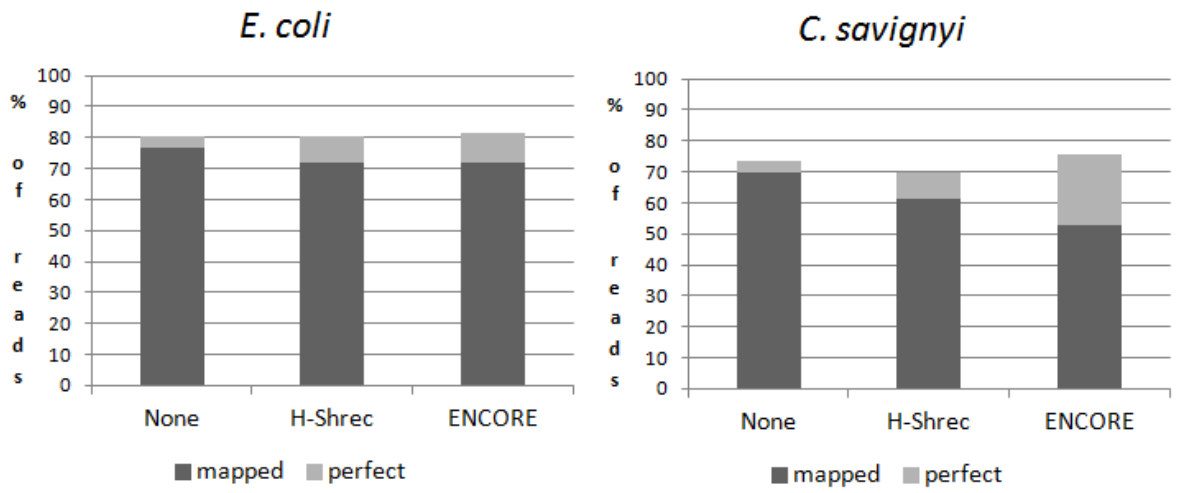

Figure 2.4: *Percentage of mapped (95% identity) and perfect (100% identity) reads before and after correction in the E. coli and C. savignyi datasets.*

In our first experiment, we use reads generated by the Roche/454 sequencing platform sampled from an antibiotic resistant *E. coli* (NCBI Short Read Archive, accession code: SRR024126). This dataset includes 110,388 reads amounting to ∼10x sequence coverage. In our second experiment, we use a subset of the real Sanger reads from the original *C. savignyi* sequencing project (see previous section). This subset is identified by mapping the reads to the 33mbp reference sequence we have compiled using the mapping software MUMmer[1] (version 3.22 [38]). A read is included in the subset if at least 90% of its sequence maps to a location on the reference with a minimum of 95% identity. For each read that is added to the subset we also include its pair, yielding a total of 558,936 reads. The total number of bases in this dataset is 358mp, which equals roughly 10x haploid coverage. Both datasets are quality trimmed before correction.

Since we do not have the ground truth in this case, we assess the performance of ENCORE by mapping the reads to the reference sequences. Results are summarized in Table 2.2. ENCORE and H-Shrec perform similarly on the *E. coli* dataset. Indeed,

[1]All mapping tasks in this section are performed using this software unless otherwise stated.

Table 2.3: *The effect of error correction on real Illumina reads.* A read is considered to be perfect if the entire read maps with 100% identity. A read is considered miscorrected if the read maps with lower identity after correction. Time is reported as wall clock time in minutes. The number of threads is set to 8 for both programs.

Correction	No. of perfect reads	Total size (mbp)	No. of mis-corrected reads	time (min)	space (gb)
None	1,260,759	146.57	-	-	
Coral	1,603,058	188.67	164,406	80	9.2
ENCORE	1,642,085	192.59	25,048	250	4.5

the probabilistic framework implemented in ENCORE might not have a substantial advantage over the coverage-based methods when a haploid and non-complex genome is considered. On the other hand, ENCORE has a noticeable advantage over H-Shrec on the *C. savignyi* dataset. For example, after correcting with ENCORE, the number of reads mapping perfectly increases by more than 6-fold, while H-Shrec only yields a 2-fold increase. Furthermore, fewer reads map to the reference at the 95% threshold after correcting with H-Shrec than without any correction, indicating that the overall gain might still be negative (Figure 2.4).

In our third experiment, we evaluate the performance of ENCORE on the Illumina platform with a comparison to Coral [64]. For this experiment, we use 160bp long Illumina reads from a recent *C. savignyi* sequencing project (data unpublished). As above, we choose a subset of these reads by mapping them to the 33mbp reference sequence taken from the draft *C. savignyi* assembly [73]. 6,527,699 reads that map with at least 95% identity are quality trimmed before correction, constituting roughly 25x coverage. We run both programs in multi-threaded mode using 8 threads. Table 2.3 summarizes the results.

We remark that this is a challenging dataset both from the perspective of error correction and of performance evaluation. The former is due to the shorter read length compared to Roche/454 and Sanger sequencing. Not only short repeats that can be resolved with long reads can no longer be resolved via sequence alignment, but also higher coverage depth is required to compensate for the shortness of the reads, increasing the computational burden. The latter is due to the fact that the Illumina reads are sampled from a different *C. savignyi* individual, for which no reference assembly exists. Thus, we perform the evaluation using the available reference sequence assembled from Sanger reads. For an organism with low polymorphism, this fact may be ignored. Nevertheless, *C. savignyi* is highly polymorphic; for example, it has an estimated SNP rate of $\sim$5% [72]. We remind the readers to consider the results of Table 2.3 under this light. On the other hand, we expect any such bias to affect both programs similarly and believe that the given comparison is still of value for this reason.

As there may be genuine differences between the reference and the reads due to polymorphism, we employ a different evaluation strategy for this dataset. In addition to the number of reads that map perfectly, we look at miscorrected reads, defined as reads that map with lower identity after correction. Since this dataset is computationally more challenging that the previous datasets, we also report the time and memory statistics for each program.

From Table 2.3, we see that both ENCORE and Coral only slightly increase the number of perfectly mapping reads. Note that these numbers are lower bounds, since a portion of correct reads will not map perfectly to the reference due to genuine SNPs and indels. On the other hand, similar to our previous experiments, we see that ENCORE corrupts fewer reads than Coral. Despite being slower than Coral, ENCORE requires less memory due to its efficient k-mer indexing strategy.

Parameters

To run H-Shrec (Version 1.0), we use the largest strictness value the program accepts for each dataset. For the *C. savignyi* datasets, we set the number of iterations to 1 (more iterations introduced more errors). For the *E. coli* dataset 3 iterations are used. The other parameters are left at defaults. To run Coral (Version 1.3), we used the default parameters for Illumina. For the Sanger and Roche/454 datasets, ENCORE is run with an error threshold of 0.07 and minimum overlap size of 30bp (k-mer size set to 13). For the Illumina dataset, the error threshold is set to 0.04 and the minimum overlap size to 45bp (k-mer size set to 14). We run Coral and ENCORE in parallel mode using 8 threads.

2.4 Discussion

Despite being motivated by the lack of error correction algorithms suitable for highly polymorphic genomes, the methods implemented in ENCORE are suitable for a variety of sequencing projects. The use of multiple sequence alignments ensure that ENCORE is applicable to several different sequencing platforms with divergent error landscapes. Moreover, combining quality scores with an efficient probabilistic framework allows us to reliably correct errors when sequence coverage is low or variable. In contrast, methods that do not use quality scores and/or heavily rely on coverage are not effective on such datasets.

ENCORE also adapts well to the nature of the datasets at hand. For example, it performs well on a bacterial genome using the same parameters as for a diploid genome with a very highly polymorphism rate. In particular, datasets with low or no polymorphism may still benefit from the Naive-Bayesian framework since repeats, while discarded by most other methods, are handled in a similar way to alleles.

2.4.1 Effect of the independence assumptions

Naturally, Naive-Bayes models work better if the independence assumptions are justified although they have been shown to work reasonably well in a variety of applications for which these assumptions are severely violated [18]. In our application, the conditional independence assumptions are partially justified. At the higher level, the evidence of a sequencing error at a particular base of a read is expected to have no effect on the probability that another read has an error at the same location since sequencing errors are believed to be mostly independent. Note that this excludes indel type errors in the Roche/454 platform, however, we handle indels separately and do not assume independence in this case.

At the lower level, we make a less justifiable assumption: that the bases of two reads are independent if they are not true neighbours. Yet, our set of pairwise alignments is non-random: even if two reads belong to highly divergent haplotypes, we expect a large fraction of their bases to be common. Nonetheless, the use of a high prior probability seems to help counter-balance this violation. In addition, we only use the posterior probability of two reads being neighbours as a weighing factor in calculation of the consensus nucleotide, which perhaps does not require a precise estimation.

Chapter 3

Polymorphic genome assembly

3.1 Introduction

The emergence of high throughput sequencing platforms enabled researchers not only to study established model organisms in depth via re-sequencing projects but also to work on new model organisms via *de novo* sequencing. Although some biological problems can be investigated using alternative sequencing techniques such as transcriptome sequencing (a.k.a. RNA-seq), none of these methods provide as complete information as whole genome sequencing.

Since the old capillary sequencing technology was expensive and time consuming, until recently, sequencing projects were reserved for a limited number of species. In addition to humans, these included model organisms such as mouse or fruit fly; agriculturally important plants such as corn; and prokaryotes such as virus and bacteria. These species typically have low levels polymorphism. For example, model organisms and agricultural plants are often inbred, which can drastically reduce the polymorphism level within the population over sufficient time.

With the cost of sequencing in decline and throughput in rise, projects such as Genome 10k [1] are already underway to sequence wild-type individuals from many diverse species.

As we have discussed in the first chapter, there is growing evidence that some of these organisms might be more challenging to assemble than the ones sequenced to date due to elevated polymorphism levels.

Not unlike the error correction tools we have seen in the previous chapter, current assembly tools are adapted to polymorphism levels (e.g. $< 1\%$ SNP rate) observed in humans and other model organisms. When confronted with highly polymorphic data, these tools require substantial tailoring and/or manual intervention [76, 73]. Assembling such genomes, already difficult with the more reliable Sanger sequencing, is a challenge even greater with the new sequencing platforms. If we are to make sense of the large volumes of sequencing data produced daily in numerous sequencing centers around the world, this is a challenge worth addressing.

Motivated by this notion, we devote this chapter to genome assembly for organisms with high polymorphism rates. In particular, we limit our attention to *de novo* assembly of such genomes. Although reference-guided assembly is an important research direction, re-sequencing of these genomes is only marginally easier than *de novo* efforts. This is due to the fact that reliable mapping of reads to the reference is difficult. Moreover, structural variations such as large indels and inversions may cause misassemblies, further complicating downstream analyses.

In the next section, we summarize common approaches to genome assembly. Then, we give an outline of our assembly algorithms followed by detailed explanation of these algorithms. We evaluate our method on two different genomes and end the chapter with a discussion.

3.1.1 Related work

Genome assembly has been a topic of interest since the introduction of the first generation sequencing technologies [65]. Earlier genome assemblers built for Sanger reads have typically employed either greedy heuristics [75, 26] or a framework referred to as Overlap-

Layout-Consensus (OLC) [52, 3, 10, 51]. As the name suggests, the basic OLC framework has three main stages. In the first stage, the reads are compared to each other to determine which read pairs have overlaps. The next stage involves building a graph where the reads serve as nodes and the list of overlaps serve as edges. This graph, named the *overlap graph*, is then subjected to various procedures such as error removal and repeat resolution. In the final stage, a consensus sequence is computed using the reads along a chosen path in the graph. Each consensus sequence is called a *contig*. The OLC assemblers vary by how they implement each stage, with the second stage typically being at the core of these differences.

Originally motivated by the work on Sequencing-By-Hybridization (SBH) [28], an alternative assembly approach is based on a structure called the *de Bruijn Graph* (DBG). Proposed as a new sequencing strategy, SBH involved synthesizing all possible DNA molecules of a small fixed length and observing the hybridization of these molecules with the target genome. The outcome of this experiment would therefore be the knowledge of all k-mers contained in the genome. SBH was ultimately abandoned since the experiment is impractical except for very small values of k. However, its theory inspired the DBG assembly approach.

The DBG method for regular sequencing reads starts by computationally breaking down the reads into their k-mers, mimicking an SBH experiment. These k-mers are used to build a graph where each k-mer represents an edge between two nodes, denoting the ($k-1$ long) suffix and prefix of the k-mer. Pevzner *et al.* [55] has suggested finding a Eulerian path - a path that visits each edge exactly once - in this graph as a solution to the assembly. This is motivated by the fact that the true assembly must include all the k-mers found in the reads as a subsequence. Although efficient algorithms exist for finding a Eulerian path in an arbitrary graph, this path need not be unique. In fact, an exponential number of distinct Eulerian paths may exist in general. A proposed remedy for this problem is to use the reads to guide this path and ensure the Eulerian path thus

generated is consistent with the read set. However, this formulation makes the problem considerably harder to solve [47].

Initially, the DBG approach was not widely adopted in part due to its sensitivity to sequencing errors and the fact that a substantial amount of information is lost by decomposing the reads into smaller fragments. With the development of high throughput sequencing technologies, there is a revived interest in this approach. First, less information is lost by breaking down the short reads produced by these technologies into their k-mers. Second, these sequencing technologies typically produce coverage depths much higher than possible with the capillary sequencing. This increased data size is problematic for the overlapping stage of the OLC approach. In contrast, the DBG methods avoid this time-consuming step using the exact matches between k-mers.

DBG-based assemblers developed for short reads implement different methods than the Eulerian path method [79, 8, 9, 70, 43]. Like OLC assemblers, these assemblers mainly differ by how they handle errors and repeats, and if applicable, how they treat paired reads. For instance, EULER-USR [9] has a step to detect errors in k-mers before graph creation whereas Velvet [79] performs error correction directly on the graph. On the technical side, AbySS [70] uses a Message Passing Interface protocol to distribute the memory over a computing cluster, while SOAPdenovo [43] works by performing multiple passes on compressed data structures.

Although recent work on genome assembly has been dominated by the DBG approach, a number of assemblers developed for high throughput sequencing apply different techniques. For example, SSAKE [77] and SHARCGS [17] are based on greedy heuristics, while Edena [25] adapts the OLC approach to short reads by computing only exact overlaps with the help of a suffix array. A similar approach is adapted by Simpson and Durbin [69], who build a Ferragina-Manzini(FM)-index to compute overlaps efficiently. Like Edena, this approach can only detect exact overlaps and has a large memory footprint.

As we have discussed at the beginning of this chapter, most of the tools we mention above ignore polymorphism in their methods. The assemblers specially developed for short reads are not very successful in assembling large and complex genomes, even when these genomes have low polymorphism rates. Yet, highly polymorphic genomes are already difficult to assemble with the traditional Sanger sequencing. For instance, Arachne [3, 30], one of the most successful assemblers developed for Sanger sequencing, was substantially modified to assemble the highly polymorphic *C. savignyi* genome [76]. Still, the resulting assembly had lower quality than typically achieved with Sanger sequencing, prompting more effort to improve this assembly with a combination of manual and automated approaches [73].

In this chapter, we present Hapsembler, an assembler developed specially for highly polymorphic genomes. Hapsembler is based on the OLC approach with several novel additions. In particular, we introduce a structure called the *mate pair graph*. A mate pair graph is essentially an overlap graph built from read pairs instead of single reads and is very useful in resolving repeats in addition to the ambiguities caused by polymorphism. Although the idea of using paired reads for repeat resolution is not new, most assemblers employ heuristics that only use local information [8, 80] while the mate pair graph provides a unique way of globally representing the information contained in read pairs. An exception to this is "paired de Bruijn graphs" introduced by Medvedev *et al.* [48], which incorporate the mate-pair information directly into the graph. The key difference of the mate pair graph from the paired de Bruijn graph is that the mate pair graph requires the existence of a path to connect two mate pairs, while the latter ignores whether such a path can be found or not. Furthermore, the paired de Bruijn graph approach is very sensitive to insert size deviation and its performance rapidly deteriorates with large insert sizes.

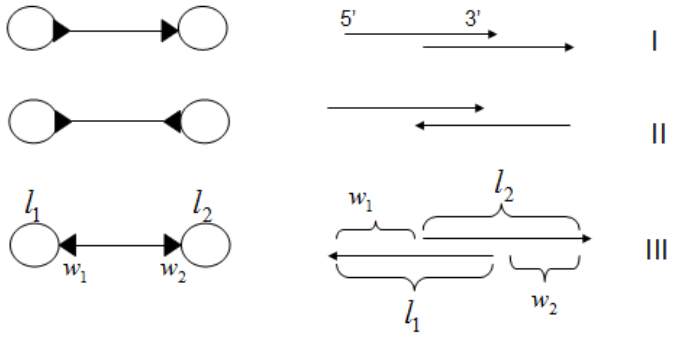

Figure 3.1: *Representation of read overlaps as a bidirected graph.* The directed lines to the right represent the reads where the arrowed end is the 3' end while the flat end is the 5' end. The node weights, l_1 and l_2, are equal to the lengths of the reads. The edge weights, w_1 and w_2, denote the lengths of the read portions that are not covered by the overlap.

3.1.2 Hapsembler

Here we give a brief outline of Hapsembler's algorithm. As an OLC assembler, the first step of Hapsembler is overlap computation. This step is performed using the methods explained in Section 2.2.1. Next, an overlap graph is built using the list of computed overlaps. The overlap graph is then subjected to various graph simplification procedures. In the next step, this simplified graph is used to build a mate pair graph, which is subjected to similar simplification procedures. In the final step, consensus sequences generated from maximal paths in the mate pair graph are output as contigs. The following sections explain these steps in more detail.

3.2 The overlap graph

Since the overlap graph forms the basis of a typical OLC-based assembler, in this section we describe this structure in detail. In general, an overlap graph is a graph where the nodes represent reads and the edges represent the existence of overlaps between the

corresponding reads. In our implementation, we form the overlap graph as a *bidirected graph* (Figure 3.1). Formally, a bidirected graph G is a graph where each edge can acquire either of the two types of arrows at each node; *in-arrow* and *out-arrow*. As a result, there are 3 possible types of edges in a bidirected graph; *in-out*, *out-out* and *in-in*. A valid walk in a bidirected graph must obey the following rule: If we come to a node using an in-arrow we must leave the node using an out-arrow. Similarly, if we come to a node using an out-arrow we must leave using an in-arrow. In the former case, the node is said to be *in-visited* and in the latter case it is said to be *out-visited*.

There is a direct correlation between bidirected edges and the possible ways two double-stranded DNA sequences can overlap (Figure 3.1). Representing read overlaps with a bidirected graph has the advantage that reverse complements of the reads are automatically handled. To help estimate the length of the DNA sequence that is spelled by a walk in the overlap graph, we assign weights to the nodes and edges as illustrated by Figure 3.1.

In fact, there are two more types of read overlaps, omitted in Figure 3.1: overlaps caused by reads that are entirely contained within other reads. We refer to such reads as *contained reads*. These overlaps, if they were to be represented in a bidirected graph, would require two edges. Furthermore, part of the weights associated with these edges would have to be negative in order to obey the edge weight definition. This is undesirable from the perspective of some of the graph algorithms we apply on this graph. Consequently, we discard contained reads before building the overlap graph.

Transitive edge reduction

Many edges in the overlap graph are redundant since they can be inferred by other edges. These edges can be removed from the graph using an operation called *transitive edge reduction* [51] as illustrated in Figure 3.2. Many OLC assemblers adapt this operation as it significantly reduces the size of graph, saving memory and computation time.

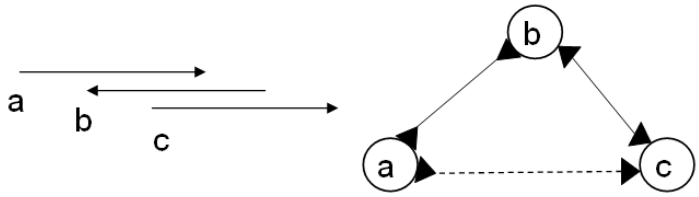

Figure 3.2: *Transitive edge reduction.* Since we can reach the node **c** from **a** via **b**, we do not need the edge between **a** and **c**. During this procedure, we check whether the two paths have the same overall length within a permissible difference f, where f is defined as the total number of indels that are present in the pairwise alignments associated with the overlaps. Otherwise, the edge is not deleted.

We remark that transitive edge reduction has another, subtle effect on the overlap graph. In an ideal environment, the initial overlap graph built as above is guaranteed to have a Hamiltonian path, which is a path that visits every node exactly once. Such a path will also represent a solution to the assembly problem: the DNA sequence spelled by this path will contain every read as a subsequence. Transitively reducing the edges voids this guarantee. On the other hand, unlike the Eulerian path problem, finding a Hamiltonian path in an arbitrary graph is NP-complete even if we know such a path exists. Thus, in practice, the apparent information loss due to this operation is not important.

Graph pruning

Typically, after transitive edge reduction, a majority of the nodes in the overlap graph have one incoming and one outgoing edge. Indeed, if all the reads are error-free and the genome contains no repeats longer than the minimum overlap size, this operation should reduce the overlap graph to a single path. In practice, the reduced graph often contains a large number of nodes with higher degrees depending on the sequencing error rate and the repeat structure of the genome.

Although repeats are generally harder to resolve at this stage, most graph-based

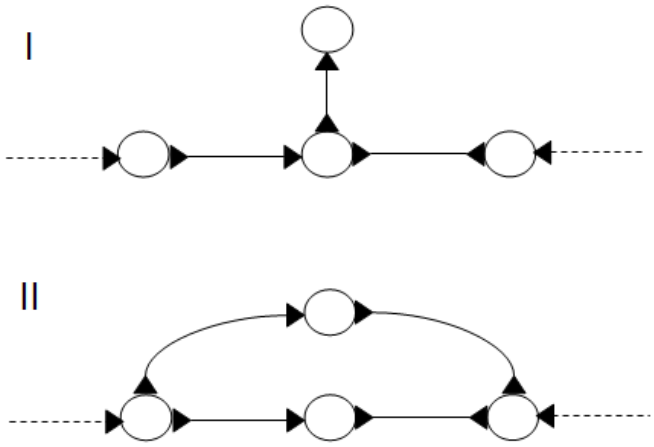

Figure 3.3: *Sprouts and bubbles.* **Top:** A sprout is identified by a short dead-end path (often a single node) splitting from a longer path. **Bottom:** A bubble is a short alternative path, which eventually converges to the main path.

assemblers implement methods to remove the artifacts caused by sequencing errors. In our case, these artifacts appear in two forms: sprouts and bubbles (see Figure 3.3). Sprouts are usually caused by errors that are concentrated towards the end (or beginning, as is common in Sanger sequencing) of a read. For such reads, no overlap is found on one end although the other end is connected to the graph. Sprouts can be easily identified as short dead-end paths splitting from a main path. In theory, sprouts may also be caused by coverage gaps. However, such cases must be rare since the coverage gap should be located exactly after a repeat boundary in order to appear as a sprout. We remove sprouts if they are below a length threshold. Bubbles are usually caused by errors that appear in the middle of reads, although they may also be caused by SNPs and short indels. We remove bubbles only if the two sequences have similar lengths shorter than a threshold.

3.3 The mate pair graph

As we have discussed in the first chapter, an additional source of information in genome assembly is paired reads. While generating reads with a few thousand base pairs is beyond the reach of available sequencing technologies, paired reads can be generated from significantly larger fragments. For example, large scale sequencing projects using Sanger sequencing typically include 20-40kbp insert libraries as part of their datasets. Generating such inserts with the second generation sequencing technologies is currently more difficult; however, paired reads with relatively long insert sizes are routinely available. Since these inserts can span longer repeats than the reads, paired reads have a substantial advantage over single reads. Though many assemblers use paired reads to resolve repeats and generate longer contigs, we believe that their potential has not been sufficiently explored.

Our strategy is inspired by the following, seemingly utopian, idea: *Treat every read pair as one single read.* Surprisingly, this idea is in fact applicable in the context of an overlap graph. In explanation, it is possible to build an "overlap graph" of such artificially constructed reads.

Recall that, when discussing transitive edge reduction, we have stated that if the genome has no repeats longer than the minimum overlap size, this operation should reduce the overlap graph to a single path, representing the genome. If we could treat read pairs as single reads, we could set the minimum overlap size to a value greater than the read length, potentially resolving much longer repeats.

Naturally, there are several caveats in treating read pairs as single reads. First, we only have an estimate for the size of the genomic region between a read pair and more importantly we do not know the sequence of this region. These obstacles prevent us from building the true overlap graph of paired reads; we can, however, build a graph that contains the true overlap graph as a subgraph. We call this graph the *mate pair graph* and explain how to build it below.

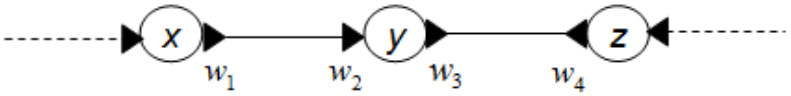

Figure 3.4: *Calculation of path lengths.* The length of the path from x to z is calculated as $w_1 + w_3$, while the length from z to x is calculated as $w_4 + w_2$.

3.3.1 Definitions

A *mate pair*[1] is a pair of reads that has an insert size distribution with mean μ_l and standard deviation σ_l. The insert size is defined to include the read lengths. The valid range of an insert size is taken as $\mu_l \pm (k\sigma_l)$, where k is a real number controlling the largest deviation from the mean we are willing to allow.

Suppose we are given a bidirected overlap graph G built from a set of reads. For two nodes u, v in G, we use $indist(u, v)$ to denote the length of the shortest path from u to v, which enters v using an in-arrow. Similarly, we use $outdist(u, v)$ to denote the length of the shortest path from u to v, which enters v using an out-arrow. The length of a path is calculated as a sum of the corresponding edge weights (Figure 3.4). For a node $v \in G$, $length(v)$ denotes the length of the corresponding read. We assume the correct orientation of the reads in a mate pair to be forward-reverse, which corresponds to a path that leaves both nodes using an out-arrow.

A mate pair graph H is defined as a bidirected graph where the nodes represent mate pairs and the edges represent the existence of overlapping paths between two mate pairs.

3.3.2 Finding mate pair paths

To build H, we first have to identify all paths of length $\mu_l \pm (k\sigma_l)$ between each mate pair in G. In general, there may be an exponential number of such paths in the graph. However,

[1] In this chapter, we use this term to refer to generic paired reads regardless of the lab technique used to generate them.

we can identify the subgraph of G which contains all paths with length $\leq \mu_l + (k\sigma_l)$ between two nodes in polynomial time.

This idea can be summarized as follows. Let nodes a and a' be a mate pair in G. First, we perform Dijkstra's [16] shortest path finding algorithm starting from a (and leaving the node using an out-arrow). While Dijkstra's algorithm is originally invented for directed or undirected graphs, the generalization to bidirected graphs is straightforward. The only difference is that instead of a single distance from the source, we have to keep track of the shortest in-distance and out-distance separately for each node. We also modify the algorithm so that only the nodes that are within a distance of $\mu^* = \mu_l + (k\sigma_l)$ are enqueued.

During this search, if we do not encounter a' it means there is no path in the graph between a and a' less than the given length. This situation can arise for several reasons: (1) there might be a coverage gap between the mates, (2) the insert size deviation might be higher than we allow, (3) we might be missing overlaps (due to sequencing errors, short overlaps, etc) or (4) the mate pair can be chimeric, meaning that the reads are in fact unrelated. The latter is due to a sequencing anomaly and is only common in certain insert library generation techniques. Typically, a majority of these cases are explained by the first scenario.

If we encounter a' during the search, we do another pass of Dijkstra's, this time starting from a'. During this second pass, we enqueue a node if and only if the sum of its shortest distance from a, its current distance from a' and the read's length is less than μ^*. Furthermore, we put such nodes into a set which we call $S_{aa'}$ together with their shortest distances from a and a'.

After this second pass, we end up with a set of nodes, $S_{aa'}$, that are guaranteed to lie on at least one path that has length less than μ^* between a and a'. This set is also exhaustive; that is, all nodes v that satisfy $indist(a, v) + outdist(a', v) + length(v) < \mu^*$ or $outdist(a, v) + indist(a', v) + length(v) < \mu^*$, are included in $S_{aa'}$. Note that we find

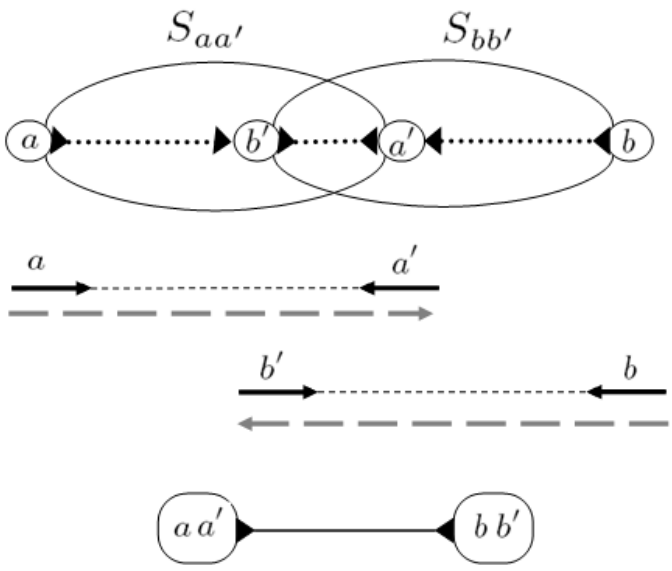

Figure 3.5: *Finding overlapping mate pairs.* By comparing the sets $S_{aa'}$ and $S_{bb'}$, we can find whether the two mate pairs have overlapping paths. If such a path exists, we create a bidirected edge between aa' and bb'. The direction of this edge is based on the (arbitrarily) assigned direction for each mate pair and how they overlap.

this set of nodes in polynomial time even though there might be an exponential number of paths between a and a'.

3.3.3 Detecting overlapping paths between mate pairs

The process described above is repeated for each mate pair, yielding a collection of sets $\bar{S}$. We then use these sets to detect overlapping paths between mate pairs. Consider two mate pair sets $S_{aa'}$ and $S_{bb'}$. To decide if these mate pairs have paths that overlap with each other, we first check whether the following conditions hold:

$$a \in S_{bb'} \tag{3.1}$$

$$a' \in S_{bb'} \tag{3.2}$$

$$b \in S_{aa'} \tag{3.3}$$

$$b' \in S_{aa'} \tag{3.4}$$

Whenever there are less than two positive answers to these checks, an overlap of paths is not possible. Otherwise, we check if the length and orientation of the paths are compatible. For example, if we find that $a' \in S_{bb'}$ and $b' \in S_{aa'}$, we check whether the following inequalities hold:

$$indist(a, b') + outdist(a', b') + length(b') < \mu^* \tag{3.5}$$

$$outdist(b', a') + indist(b, a') + length(a') < \mu^* \tag{3.6}$$

where μ^* is defined as above. If these inequalities hold, we create a bidirected edge between the nodes aa' and bb' in H as illustrated in Figure 3.5.

This algorithm may be problematic in terms of memory for large insert sizes since we store a set proportional to the size of the insert for each mate pair. In practice, we use a slightly different version of this algorithm which can be implemented in linear space complexity independent of the insert size. In this version, we perform two extra Dijkstra's starting from each end of a mate pair, this time in opposite directions (i.e. leaving the node using an in-arrow). As before, we only enqueue nodes that are within the distance cutoff. This gives us two additional sets $S_{\hat{a}}$ and $S_{\hat{a'}}$. Then for each node b in $S_{aa'}$ we check if its mate pair b' is in $S_{\hat{a}}$ or $S_{\hat{a'}}$ depending on the orientation of the node. If the path lengths are compatible with the insert size then we put an edge between aa' and bb' in the mate pair graph. Since we can immediately determine which nodes aa' should be connected to, we do not have to store the set $S_{aa'}$ after we process aa'. Hence, this alternative algorithm takes only linear space in the number of nodes.

If the overlap graph is not missing any genuine read overlaps between the reads and there are no coverage gaps, the mate pair graph as built above is guaranteed to contain all the genuine mate pair overlaps. Nonetheless, like the overlap graph, the mate pair graph will typically contain a number of false edges due to repeats and polymorphism. Analogous to an overlap graph, the degree to which the repeats and polymorphism can be resolved is governed by the size of overlaps. By default, the minimum overlap size of the mate pair graph is the read length. A larger value can be set if the coverage depth is sufficient to ensure that the resulting mate pair graph is connected.

3.3.4 Processing the mate pair graph

The mate pair graph is structurally very similar to the overlap graph; hence some of the procedures we apply on the overlap graph can also be applied to the mate pair graph.

We have previously discussed the effect of contained reads on the overlap graph in Section 3.2. Contained mate pairs have a similar negative effect on the mate pair graph. A contained mate pair is defined as a mate pair whose sequence is entirely spanned by another mate pair. Since we do not know the real DNA sequence between a mate pair, contained mate pairs can not be detected as easily as contained reads. However, we can detect potentially contained mate pairs with the following method. Let aa' be a mate pair. We first check the sets $S_{\hat{a}}$ and $S_{\hat{a'}}$ to see if there is any mate pair bb' such that $b \in S_{\hat{a}}$, $b' \in S_{\hat{a'}}$ and:

$$outdist(a, b) + dist(a, a') + outdist(a', b') < \mu^* \tag{3.7}$$

where $dist(a, a')$ is the shortest distance between the pair aa' as computed during Dijkstra's algorithm. If there is any mate pair satisfying this property, aa' may be a contained mate pair so it is removed from the graph.

After contained mate pairs are removed, we perform transitive edge reduction on the mate pair graph. When creating the mate pair graph, three values are stored with each

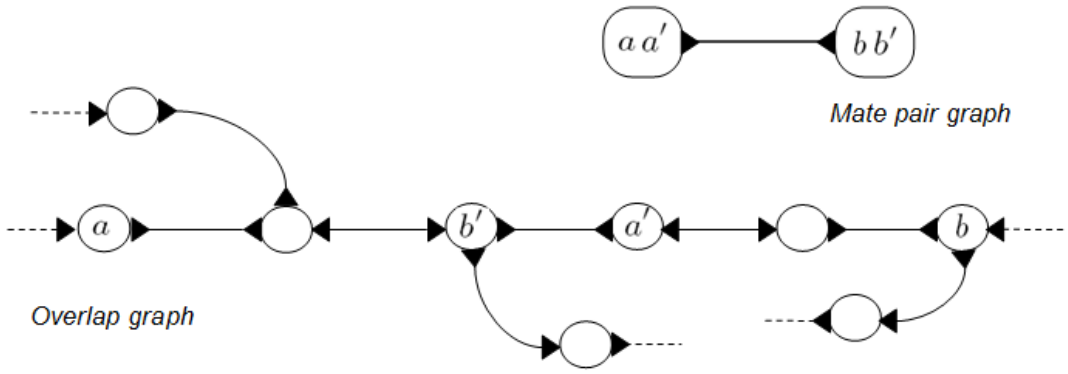

Figure 3.6: *Walking the overlap graph using mate pair edges.* Every edge of the mate pair graph corresponds to particular paths in the overlap graph. For instance, to go from the mate pair node aa' to the node bb' above, we first traverse the path from a to b', then from b' to a' and finally from a' to b.

edge. For example, for the mate pair edge between aa' and bb' of Figure 3.5, we store the distances $indist(a, b')$, $outdist(b', a')$ and $indist(b, a')$. Given three mate pair nodes where each node is connected to the other two, we decide whether one of the edges can be inferred from the other two using these distances.

3.4 Contig and consensus generation

Like the overlap graph, after transitive edge reduction, most of the nodes in the mate pair graph have one incoming and one outgoing edges, forming long non-branching paths. Using these paths as guides, we generate walks on the overlap graph which form our contigs. This process is illustrated in Figure 3.6.

Each walk on the overlap graph produces an ordered list of reads, corresponding to a contig. To determine the sequence of a contig, we first compute a pairwise alignment between each pair of successive reads in the list. These pairwise alignments are then used to generate a multiple sequence alignment in a similar way as we described in Section 2.2.2. The main difference in this case is that each newly added pairwise alignment

uses the previous read as the base. Once the multiple sequence alignment is generated, we assign the consensus nucleotide for each base via a majority voting weighted by the quality scores.

In general, these walks may not cover all the nodes of the overlap graph. For example, a portion (or *all*) of the dataset may consist of single reads. To utilize these reads, we also report any non-branching path in the overlap graph that has not been covered by a walk. These contigs are then processed in the same way as described above.

3.5 Complexity

The time complexity of building the overlap graph is the same as overlap computation, which we have discussed in Section 2.2.1. The complexity of transitive edge reduction is discussed in detail by Myers [51]. The graph pruning operations are linear in the size of the transitively reduced overlap graph, which we assume to be $O(n)$, where n is the number of reads.

The most time consuming step of building the mate pair graph is generating the mate pair sets. In theory, each mate pair set requires $O(n \log n)$ time, as this is the time complexity of Dijkstra's shortest path algorithm. As a result, the asymptotic time complexity of building the mate pair graph is $(n^2 \log n)$. In practice, however, the number of nodes traversed by Dijkstra's algorithm is limited because of the distance cutoff. Thus, the upper bound of $O(n \log n)$ is almost never reached. Furthermore, this step can be parallelized, as each mate pair set can be computed separately.

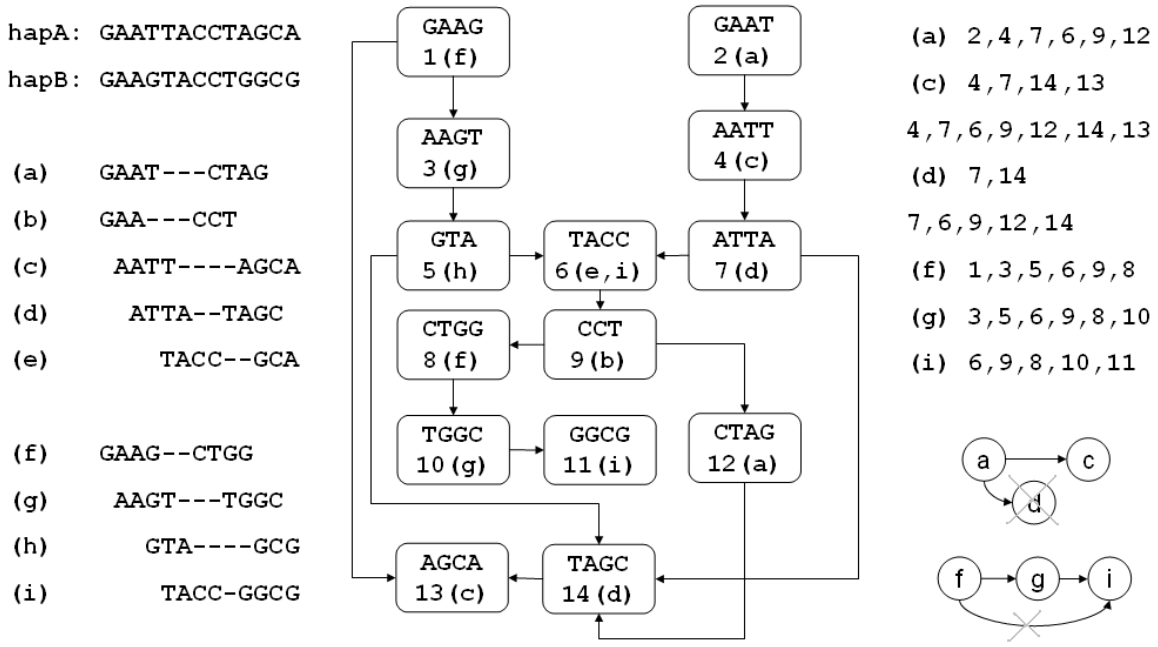

Figure 3.7: *Polymorphism and repeat resolution.* **Left:** A diploid genome and paired reads sampled from it. We do not know the exact distances between the pairs but we assume that we are given an upper bound (in this case 13bp). **Middle:** The overlap graph after removal of contained reads (i.e. GAA, GCA and GCG) and transitive edge reduction. The nodes are labelled with the mate pairs they belong to and arbitrarily numbered. The minimum overlap size is set to 2bp. **Right:** The paths between the mate pairs shorter than the given upper bound and the resulting mate pair graph. In practice, we do not need the exact paths and we only compute the set of nodes that lie on at least one path. Node d is removed since it is contained by node c. In addition, the edge between f and i is removed during transitive edge reduction. The resulting paths correspond to the two haplomes.

3.6 Polymorphism and repeat resolution with the mate pair graph

In essence, polymorphism resolution is very similar to repeat resolution and the mate pair graph is designed to exploit paired reads to handle both problems. Figure 3.7 illustrates this with a toy example. In this example, we have a diploid genome[2] which has several SNPs. After the overlap graph is built and simplified, we have several ambiguous nodes (i.e. nodes with three or more edges). Some of these ambiguities are due to short repeats and some are due to regions that are identical in both haplomes. Yet, the mate pair graph built from this overlap graph is less tangled. Indeed, the simplified mate pair graph has exactly two disjoint paths, each spelling the sequence of one haplome (i.e. haploid genome).

3.7 Evaluation

3.7.1 Assembly of *E. coli*

First, we evaluate our methods on the haploid genome of *E. coli*. For this experiment, we use the dataset explained in Section 2.3.3. Since this dataset consists of single reads, we artificially pair the reads as follows. Reads are mapped to the reference sequence and sorted by their mapping positions allowing duplicate mappings. For each read mapping to the forward strand, an unpaired read mapping on the opposite strand that has distance closest to 8000bp is taken. If there is no such read with distance 8000±2400, then the read is left unpaired. This mapping yields 33,160 pairs with insert size mean and standard deviation of 8534.67 and 713.35 respectively. The rest of the reads are left as single reads.

On this dataset, we compare Hapsembler with Velvet [79] and Euler [9]. Table 3.1

[2]For simplicity of the example, we pretend the genome is single stranded.

Table 3.1: *Assembly of E.coli with Roche/454 reads.* Only contigs longer than 500bp are reported. Coverage and accuracy are computed by mapping contigs to the *E. coli* reference sequence (4.6mbp) using MUMmer [38] with default parameters. N50 is defined as the largest contig size such that the sum of contigs at least as long is greater than half the genome size. "Velvet (ec)" contains the results achieved using Velvet when the reads are corrected with ENCORE.

	N50 (kbp)	No. of contigs	Total size (mbp)	Coverage (%)	Accuracy (%)
Unpaired					
Hapsembler	72.4	128	4.6	90.3	98.6
Velvet	41.2	199	4.5	89.5	98.4
Velvet (ec)	72.3	126	4.6	90.6	98.5
Euler	8.1	913	4.7	88.6	98.6
Paired					
Hapsembler	103.7	111	4.6	90.9	98.6
Velvet	41.9	189	4.6	89.5	98.2
Velvet (ec)	72.3	132	4.6	90.6	98.5
Euler	9.4	765	4.5	88.4	98.6

Table 3.2: *Running times in seconds taken by the assemblers on the E. coli dataset.* The read correction time taken by ENCORE is reported separately. Since ENCORE and Hapsembler are multi-threaded, we give the wall-clock time and report the total CPU time in paranthesis. 16 threads are used in each case. For the other tools only the CPU times are reported.

	ENCORE	Hapsembler	Velvet	Velvet (ec)	Euler
Unpaired	65 (422)	50 (254)	106	74	54
Paired	-	67 (241)	104	75	54

shows the results of two experiments. In the first experiment, no pairing information is supplied to the assemblers. In the second experiment, we use the artificial pairings described above. For Velvet and Euler, we report the results using the k-mer size that achieves the best N50 value (19 and 23 respectively). For Hapsembler, the overlapping parameters are set to the values used for error correction (see Section 2.3.3).

As part of its pipeline, Hapsembler corrects the reads using the ENCORE algorithm (see Section 2.2) before assembling them. To assess how much of Hapsembler's performance is due to error correction, we also report the results achieved by Velvet using these corrected reads. For the unpaired case, we see that Hapsembler and Velvet perform very similarly if Velvet is supplied with corrected reads. Indeed, with single reads, Hapsembler's ability to resolve repeats is limited. For the paired case, Hapsembler performs better than Velvet even when both programs use corrected reads, demonstrating the utility of the mate pair graph in resolving repeats. Table 3.2 shows that when multiple cores are available the running time of Hapsembler is comparable to Velvet and Euler on this dataset.

Table 3.3: *Assembly of C.savignyi with Illumina reads.* Only contigs longer than 200bp are reported. Coverage is computed by mapping contigs to the diploid reference sequence (total size 336mbp) using MUMmer with default parameters. N50 is calculated as the largest contig size such that the sum of contigs at least as long is greater than half the size of the total reference sequence. For SOAPdenovo and AbySS, we also report the results (represented by "ec") achieved by these tools when the reads are first corrected with ENCORE.

	N50 (bp)	Largest contig (bp)	No. of contigs	Total size (mbp)	Reference coverage (%)	Haplome specificity (%)
Hapsembler	1,970	39,816	182,064	285.54	79.74	31.24
SOAPdenovo	568	30,516	472,203	274.17	66.49	27.89
SOAPdenovo (ec)	685	30,516	433,082	279.79	68.07	29.73
AbySS	368	28,653	533,712	246.69	60.46	20.86
AbySS (ec)	380	37,442	536,057	250.83	60.94	20.99

3.7.2 Assembly of *C. savignyi*

To evaluate Hapsembler on a highly polymorphic genome, we use *C. savignyi*. As our dataset, we use 87 million paired-end Illumina reads from a *C. savignyi* re-sequencing project (data unpublished). The reads are 160bp in length with an estimated insert size of 600bp. On this dataset, we compare Hapsembler to AbySS [70] and SOAPdenovo [43]. We choose these DBG-based assemblers for their relative efficiency in handling large sequencing datasets.

The results are given in Table 3.3. For AbySS and SOAPdenovo, we set the k-mer size

Table 3.4: *Running times in minutes taken by the assemblers on the C. savignyi dataset.* The read correction time taken by ENCORE is reported separately. Since ENCORE, Hapsembler and SOAPdenovo support multi-threading, we report both wall-clock and CPU times for all tools. 16 threads are used in each case.

	ENCORE	Hapsembler	SOAPdenovo	Abyss
Real	1,145	1,955	31	327
CPU	15,819	6,884	163	322

to 63.[3] The wall-clock and CPU times taken by the assemblers and the error correction module are reported in Table 3.4.

As we have discussed in Section 2.3.3, since the available reference sequence is assembled from a different *C. savignyi* individual, we note that the coverage values are expected to be low. Furthermore, contig accuracy can not be determined in the usual way due to possible recombination events between the two individuals. Instead, to evaluate the accuracy of contigs, we report the proportion of long contigs (>600bp) that map entirely to one haplome. This is reported as haplome specificity in Table 3.3.

We see that on this dataset Hapsembler produces substantially longer contigs and covers more of the reference sequence with fewer contigs. Note that Table 3.3 only reports contigs that are longer than 200bp. In reality, both AbySS and SOAPdenovo report a significant number of contigs (2.4m and 2.1m respectively), yet a majority of these are shorter than the read length (160bp). In contrast, Hapsembler already covers nearly 80% of the reference sequence with contigs longer than 200bp. Table 3.3 also demonstrates that although error correction improves the performance of AbySS and SOAPdenovo slightly, correction alone does not seem to explain the superior results

[3]Though the N50 values seem to increase very slightly as we increase the k-mer size, both programs run out of memory for larger values on our server with 80GB RAM.

achieved by Hapsembler. These results suggest that the mate pair graph strategy of Hapsembler is indeed more effective than the current state-of-the-art assemblers for highly polymorphic genomes.

3.8 Discussion

In this chapter, we summarized the current approaches to genome assembly and stated the lack of methods that can handle highly polymorphic genomes. We also presented a novel type of assembly graph that is designed to leverage the information contained in paired reads in order to resolve polymorphic regions. As our results show, this graph, which we call the mate pair graph, is also useful when assembling less polymorphic organisms due to its ability to resolve longer repeats.

Some of the methods we present in this chapter are computationally expensive. In particular, overlap computation and building mate pair sets are the two most time consuming steps. On the bright side, both steps are suitable for parallelization. Hapsembler, which implements these methods, is currently slower than the widely used genome assemblers available for high throughput sequencing. Yet, this gap is not prohibitively large. For a bacterial genome such as *E. coli*, the time requirements of Hapsembler is comparable to that of Velvet and Euler when a moderate number of cores are available. For a large and complex genome such as *C. savignyi*, Hapsembler takes considerably more time than SOAPdenovo and AbySS. On the other hand, Hapsembler is more efficient in terms of memory: while the peak memory required by Hapsembler is around 35GB for this dataset, AbySS and SOAPdenovo require significantly more memory for large k-mer sizes.

Chapter 4

Scaffolding

4.1 Introduction

In the previous chapter, we have seen that current assemblers developed for high throughput sequencing platforms can produce high quality draft assemblies for small genomes. In contrast, the assemblies produced for more complex genomes using short reads are typically very fragmented. Hence, most sequencing projects are left as draft assemblies because the finishing stage remains costly and time consuming.

The finishing stage, which requires additional sequencing and specialized methods, can significantly benefit from *scaffolding*: the process of linking contigs into larger gapped sequences using paired read information. Briefly, a contig is inferred to succeed (or precede) another contig if one end of the pair maps to the first contig and the other end maps to the second contig. Since the relative orientation of the reads in a pair is also known (eg. forward-reverse for Illumina paired-end reads and forward-forward for AB/SOLiD reads), we can also infer the orientation of these contigs with respect to each other. Furthermore, the size of the gaps between these contigs can be estimated using the insert size distribution. A chain of contigs constructed as such is called a *scaffold*.

Scaffolding can also be performed using other types of information such as the finished

genome of a closely related organism [56] or optical restriction maps [53]. In the first case, contigs from the target assembly are mapped to the genome of a closely related organism. These mappings provide information about the order and orientation of the contigs. In the latter case, a genome-wide map of the approximate locations of a restriction enzyme is attained using a special lab technique. This map is then used to identify the location of the contigs along the genome.

Like contig assembly, scaffolding is a computationally hard problem [27]. To simplify the problem, scaffolding is often performed as two sequential tasks: contig orientation and contig ordering, where each task is formulated as maximizing the number of satisfied paired read constraints. These formulations are still individually intractable and initial methods applied to this problem have largely been greedy [27, 56].

Although the assembly algorithm we have presented in Chapter 3 already employs paired read information, paired reads with a coverage gap in between are not utilized. Such pairs may even exist in high-coverage sequencing projects and are only useful in a scaffolding context. To complete our discussion of genome assembly, we devote this chapter to the scaffolding problem.

The rest of this chapter is outlined as follows. We first give a brief summary of different approaches to scaffolding and explain the shortcomings of these approaches. Then, we present two novel algorithms; one to solve the orientation problem and the other to solve the ordering problem. The former is based on a fixed-parameter tractable algorithm developed for the odd cycle transversal problem [59]. The latter is based on a combination of Linear Programming and a graph problem called the feedback arc set problem [21]. We evaluate these algorithms on several datasets and compare our results to those attained by several other scaffolders.

4.1.1 Related work

Most genome assemblers implement simple greedy algorithms to produce scaffolds using paired reads, though several stand-alone scaffolders are available. Some of these scaffolders such as Bambus [56] and SSPACE [7] also use greedy heuristics, while others employ more involved methods such as Mixed Integer Programming [63] or statistical optimization [13].

Non-greedy scaffolders often represent the scaffolding problem as a graph where nodes denote contigs and edges denote paired read links. SOPRA [13] partitions this graph into smaller parts and solves each using statistical optimization. MIP Scaffolder [63] partitions the graph in a similar way; however, it solves these subgraphs exactly using Mixed Integer Programming. Both of these scaffolders limit the size of the subgraphs in order to keep the algorithms tractable. Opera [24] applies a different partitioning scheme using a graph contraction procedure and solves the scaffolding problem with a fixed-parameter tractable algorithm based on a graph-bandwidth formulation.

Almost all approaches to scaffolding are based on maximizing the number of paired reads that are satisfied. This formulation implicitly assumes that paired read links are noisy and contigs are error-free. While the presence of chimeric pairs and inaccurate mapping of the reads justify the former assumption, there is no real justification for the latter. In our approach, we assume both type of errors are possible and integrate these in a single formula. In contig orientation, this formulation allows us to detect erroneous contigs and remove these instead of removing the conflicting edges incident to them. Furthermore, we solve this formulation exactly using a fixed-parameter tractable algorithm. In contig ordering, the use of a polynomial time, near-optimal graph algorithm allows us to position the contigs with Linear Programming instead of computationally expensive methods such as Mixed Integer Programming.

4.1.2 ScaRPA: Scaffolding Reads with Practical Algorithms

We implement our methods as a scaffolder named ScaRPA. As input, ScaRPA takes as a set of contigs and a file containing the mappings of a paired read dataset to these contigs. As a preprocessing step, ScaRPA filters ambiguously mapping reads and estimates the mean insert size and deviation using pairs mapping to the same contig. Then, ScaRPA assigns an orientation to each contig which discards a minimum number of paired read links and contigs using a fixed-parameter tractable algorithm. In the next step, these contigs are given a pairwise-consistent order which maximizes the paired read constraints using a near-optimal polynomial time algorithm. In the final step, contig positions are adjusted using a Linear Programming framework. The following sections explain these steps in detail.

4.2 Preprocessing

The first step of ScaRPA is to filter and analyze the read mappings, which can be produced by any mapping software. Like other scaffolders, ScaRPA discards a read pair if either of the reads maps ambiguously (i.e. has more than one optimal hit).

ScaRPA also analyzes the read mappings to adjust the mean insert size and deviation. Typically, for most sequencing projects, an estimate for the mean insert size is readily available. In some cases this estimate might be slightly off. An inaccurate mean value causes the gap sizes between the contigs to be estimated incorrectly, resulting in a genome-wide bias. For instance, if the given insert size is above the true mean, the total size of the scaffolds may exceed the genome size. If a sufficient number of contigs are long enough to allow both ends of a pair to map to the same contig, ScaRPA estimates the insert size distribution using these mappings.

Ideally, if we have a large number of pairs mapping to the same contig in the right orientation, the standard calculation of sample mean and variance should give a reliable

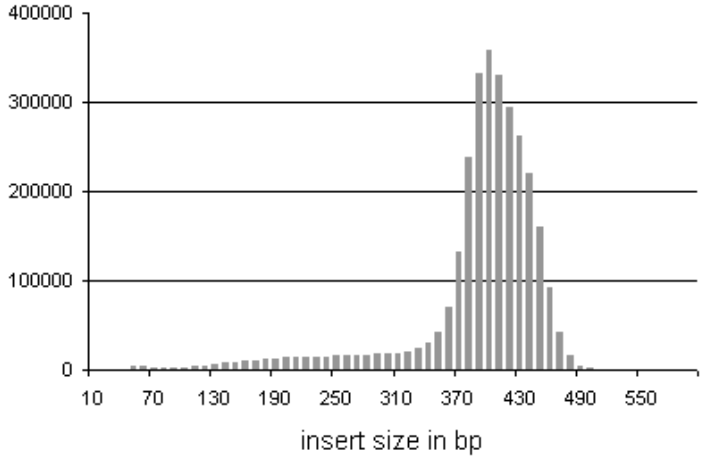

Figure 4.1: *Plotting the mapped distances between paired reads from a P. syringae dataset reveals the highly skewed shape of the library distribution.* Above, the distances are computed using pairs mapping to the same contig in the correct orientation. Mapping the reads to the reference genome instead of the contigs results in a similar plot (data not shown).

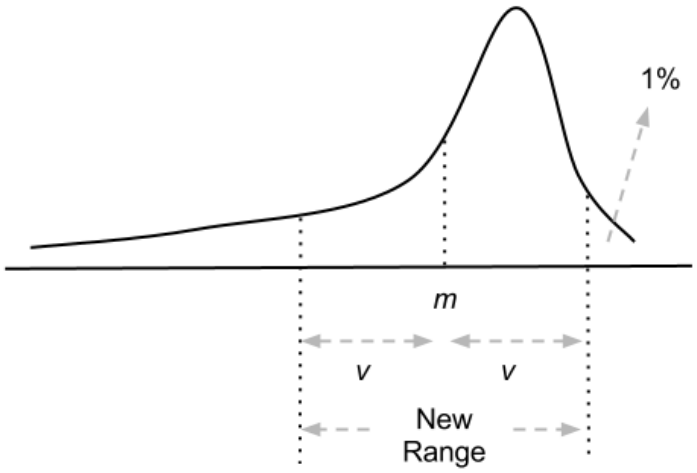

Figure 4.2: *Iterative insert size estimation.* To estimate the mean and variance of a skewed insert size distribution, we employ an iterative approach. First, we calculate the sample mean as usual (denoted with m above). Using the top percentile of the entire dataset as a cutoff, we determine a restricted range of data points ($m \pm v$). This new range is used to calculate the next mean and this procedure is iterated until convergence.

estimate. Unfortunately, real datasets, especially paired-end libraries, tend to have highly skewed distributions with a heavy tail on one side (Figure 4.1). In such cases, the sample mean and variance estimates are frequently wrong. To improve the library size statistics, we develop an iterative procedure as follows. Let m_0 be the sample mean calculated as $m_0 := \frac{1}{n}\sum_i d_i$ where d_i are mapped distances between pairs and n is the number of pairs. Let t be the point such that 99% of all d_i lie under t and set $v_0 := t - m_0$. We then set $k = 0$ and repeat the following until $|m_{k+1} - m_k| < \delta m_k$, where $\delta << 1$:

- $m_{k+1} := \frac{1}{n_k}\sum_i d_i$ such that $d_i < m_k + v_k$ and $d_i > m_k - v_k$
- $v_{k+1} := t - m_{k+1}$
- $k := k + 1$

While this procedure converges in a few iterations for distributions such as the one in Figure 4.1, it is also harmless for non-skewed distributions. Indeed, if the distribution is symmetrical, e.g. Gaussian, the procedure stops after a single iteration returning the initial sample mean.

After the library statistics are finalized, we build a scaffolding graph, where nodes are contigs and edges are paired read links between the contigs. If there are multiple links between a pair of contigs, we bundle them using the approach of [27] to form a single edge. Briefly, this method works by combining the paired read constraints using the following equations:

$$\bar{\mu} = \frac{\sum_i \frac{\mu_i}{\sigma_i^2}}{\sum_i \frac{1}{\sigma_i^2}} \tag{4.1}$$

$$\bar{\sigma} = \frac{1}{\sqrt{\sum_i \frac{1}{\sigma_i^2}}} \tag{4.2}$$

where μ_i is the estimated distance between the contigs based on the paired read link i. Here μ_i is calculated by subtracting the distance between the mapped positions and the end of the contigs from the mean insert size. Note that it is possible for this value to be negative if the end of the contigs overlap. σ_i is the standard deviation of the insert size distribution and estimated as the square root of the sample variance.

Each edge is weighted by the number of paired reads supporting the link. Edges with support lower than a threshold are discarded. Similar to library size adjustment, if we have a large sample of pairs mapped to the same contig, we derive this threshold empirically. Otherwise it can be set manually. In the former case, we set this threshold to $\frac{\log \epsilon}{\log(1-r)}$, where $\epsilon << 1$ is a small positive constant and r is the ratio of successfully mapping pairs to all pairs mapping to a single contig. A pair is said to map successfully if the orientation of the reads are as expected and the distance between the reads is less than a large cutoff.

4.3 Orientation

In general, each assembled contig is expected to lie on an arbitrary strand of the genome. The orientation stage of scaffolding attempts to orient the contigs based on the read pairs so that within each scaffold all the contigs lie on the same strand. This is illustrated in Figure 4.3. In the absence of errors, this problem has a feasible solution easily identified via a greedy algorithm. In practice, factors such as chimeric read pairs, mismapped reads and incorrectly assembled contigs make the problem infeasible.

The orientation problem is usually formulated as follows: assign an orientation for each contig so that the maximum number of paired reads is satisfied. This formulation is NP-hard [34]. This approach, taken by most scaffolders [13, 56, 63], is justified when the majority of incorrect links are due to chimeric pairs or mismapped reads. On the other hand, links that are due to chimeric pairs tend to have low support - such errors

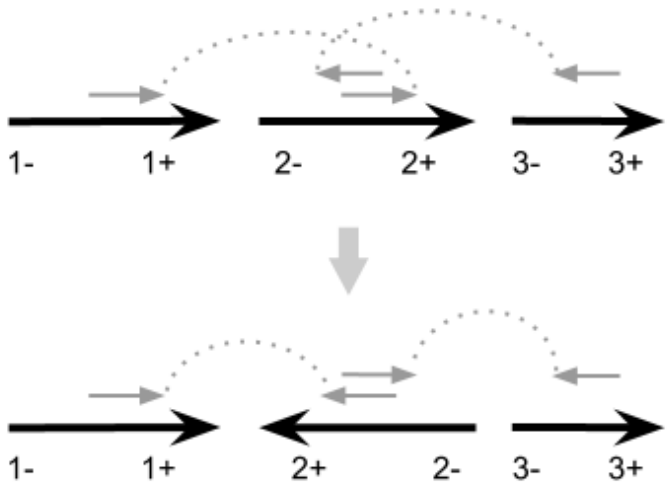

Figure 4.3: *Contig orientation.* The relative orientation of contigs with respect to each other is identified via paired read links. Here, we assume the correct orientation of a read pair is forward-reverse (i.e. paired-end orientation). If the orientation of the library is otherwise, reads are reverse complemented to match this orientation.

are expected to occur independently - and are mostly discarded during preprocessing.

We adopt a different approach and formulate the contig orientation problem as an odd cycle transversal problem. We first build an undirected graph G, where each contig c is represented by two nodes c^- and c^+ corresponding to the 5' and 3' ends of the contig. For each contig c, we add an edge between c^- and c^+. For each read pair r_1 and r_2 mapping to contigs x and y respectively, we add an edge between:

- x^+ and y^- if r_1 maps to x on the forward strand and r_2 maps to y on the reverse strand
- x^- and y^+ if r_1 maps to x on the reverse strand and r_2 maps to y on the forward strand
- x^+ and y^+ if both r_1 and r_2 map on the forward strands
- x^- and y^- if both r_1 and r_2 map on the reverse strands

Note that the contig orientation problem has a feasible solution if and only if G has no cycles containing an odd number of nodes. We exploit this property to adapt a fixed-parameter tractable algorithm developed by Reed *et al.*[59], which, given a graph G, identifies a set X of nodes with $|X| \leq k$, for any fixed k, such that $G - X$ has no odd cycles or asserts that no such set exists. In the graph theory literature, X is referred to as an odd cycle transversal. Next, we give a summary of the odd cycle transversal algorithm we adapt.

4.3.1 Finding odd cycle transversals

Algorithm 1 $OddCycleTransversal(G, k)$

Input: A graph G with vertices $\{v_1, v_2, .., v_n\}$ and an integer $k \leq n$.

Ouput: A set X of at most k vertices in G, such that $G - X$ has no odd cycles or the information that no such set exists.

Let $X \leftarrow \{v_1, v_2, .., v_k\}$

Let H be the subgraph of G induced by X

for $i = k + 1, k + 2, ..., n$:

    $H \leftarrow H + v_i$

    $X \leftarrow X + v_i$

    $X \leftarrow compress(H, X, k)$

    if $|X| > k$:

        return "infeasible"

return X

The algorithm we use to find odd cycle transversals employs a method called *iterative compression*. At each iteration, we are given as input a graph G and an odd cycle transversal of size $k + 1$ and try to construct an odd cycle transversal of size k in G. Al-

though the original algorithm [59] is presented recursively, we implement an incremental version as described in Algorithm 1.

Briefly, we start with a subgraph H of size $k+2$. Trivially, H has an odd cycle transversal X of size $k+1$. Then we call a subroutine named $compress(H, X, k)$, which decides whether H has a smaller transversal or not. If there is no such transversal, it follows that G does not have an odd cycle transversal of size k either. Hence, we terminate and return this fact. Otherwise, we proceed by adding a new vertex to H and repeat this process.

We now explain how the *compress* subroutine works. Given $H = (V, E)$ and its odd cycle transversal X, let S_1 and S_2 be two stable sets in the bipartite graph $H - X$. Using this partition, we construct an auxiliary graph H' with the vertex set $V' = V - X + \{x_1, x_2 : x \in X\}$. The edges of H' are determined as follows:

- If $e \in H - X$, we create an edge in H' between the corresponding vertices.
- If $e = (x, y)$ such that $x \in X$ and $y \in S_1$, we create an edge $e' = (x_2, y)$ in H'.
- If $e = (x, y)$ such that $x \in X$ and $y \in S_2$, we create an edge $e' = (x_1, y)$ in H'.
- If $e = (x, y)$ such that $x \in X$ and $y \in X$, we pick either of $e' = (x_1, y_2)$ or $e' = (x_2, y_1)$ and add this edge to H'.

Let $S = S_1 \cup S_2$ and let $H'[U]$ denote the subgraph of H' induced by the vertex set U. A valid partition below is defined such that $\forall x \in Y$, exactly one of $\{x_1, x_2\}$ is in Y_A. The following lemma from [59] forms the basis of the algorithm:

> **Lemma 1.** An odd cycle transversal X is of minimum size if and only if for any valid partition (Y_A, Y_B) of any subset Y of X, there are $|Y|$ vertex disjoint paths from Y_A to Y_B in $H'[Y_A \cup Y_B \cup S]$.

Using this lemma, we complete the subroutine as follows. Given the odd cycle transversal X, we test each valid partition (Y_A, Y_B) of each subset Y of X. Suppose

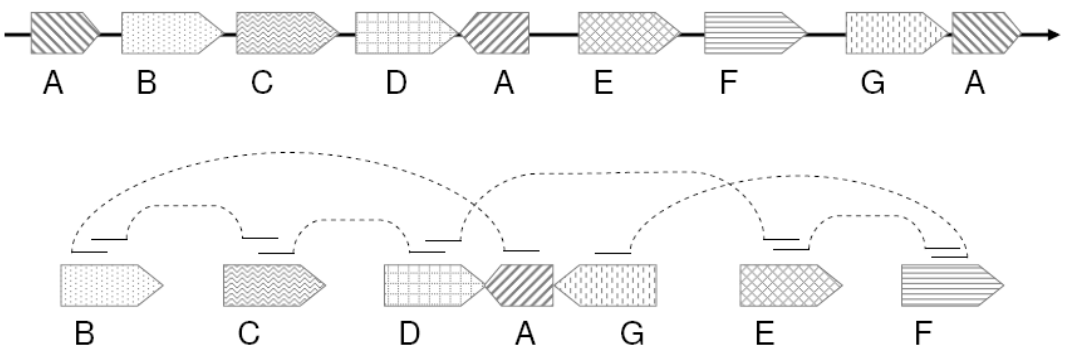

Figure 4.4: *Effect of misassembled contigs on contig orientation.* **Top:** A genome contains three copies of a repeat sequence (A), one of which has the opposite orientation with respect to the other two. **Bottom:** The genome is assembled into five contigs, including a contig that is misassembled due to the collapsed repeat. To orient these contigs consistently, at least two paired links must be discarded, whereas it is sufficient to discard one contig. In this scenario, an algorithm that only discards paired read links will produce erroneous scaffolds, while an algorithm that can discard contigs may remove the erroneous contig and produce correct scaffolds.

there are less than $|Y|$ vertex disjoint paths from Y_A to Y_B in $H'[Y_A \cup Y_B \cup S]$. Then, we can take a cutset W' separating Y_A from Y_B of size $< |Y|$. Using W', we construct a new odd cycle transversal $W \cup (X - Y)$. The set $W \subseteq H$ is determined such that, a vertex w is in W if either $w \in W'$ or $w \in X$ and at least one of its copies w_1, w_2 is in W'. Since $|W| \leq |W'| < Y$, it follows that $|W \cup (X - Y)| \leq k$.

The problem of finding the cutset W' can be formulated as a maximum flow problem with demand k, which can be solved in time $O(km)$, where m is the number of edges. We have to solve this flow problem for each choice of sets Y_A and Y_B. There are 2^{k+1} subsets of X, and for each subset Y, there are $2^{|Y|} \leq 2^{k+1}$ partitions. Alternatively, as [44] points out, this enumeration may be interpreted as all 3-way partitions of X. Consequently, $O(3^k km)$ gives a tighter bound for the subroutine. The total running time of the odd cycle transversal algorithm is therefore $O(3^k kmn)$.

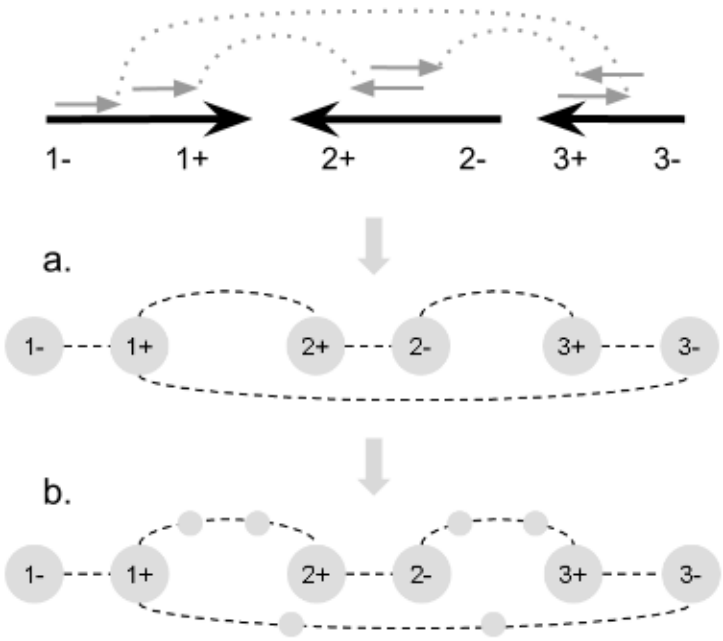

Figure 4.5: *Formulation of the contig orientation problem as an odd cycle transversal problem.* **a.** We create two nodes for each contig corresponding to the two ends of the contig and connect these nodes with an edge. Then the paired read links are used to connect the ends of the contigs. Conflicting links create odd length cycles in the resulting graph. **b.** In order to allow removal of paired read links as well as contigs, we modify the graph by adding two auxiliary nodes on each edge induced by these links. This modification preserves the odd cycles of the original graph.

The algorithm we describe above is based on removal of nodes. In some cases, it may be desirable to remove edges as well as nodes. For example, if the optimal solution involves removing multiple edges incident to a node, the contig associated with that node might have been misassembled (see Figure 4.4). In contrast, if it suffices to remove a single edge, there is a higher probability that the link is incorrect. We desire an algorithm that chooses the most sensible solution in each case.

To allow removal of edges in addition to nodes, we build another graph G', which is derived from G by inserting auxiliary nodes to some of the edges. Briefly, we insert two nodes for each edge that connects two contig nodes (but not for an edge that connects the two ends of the same contig). It is easy to see that this transformation does not alter the number of odd cycles. Any odd cycle we have in G remains an odd cycle though its size will have increased. A similar argument holds for even cycles and no new cycles are introduced. Figure 4.5 illustrates this process.

If there is a tie between discarding a contig node versus discarding an auxiliary node (representing a paired read link), we would like the algorithm to remove the auxiliary node[1]. In order to encourage the algorithm to remove paired read links before removing contigs, we order the nodes of G' such that the auxiliary nodes are considered before any contig node.

To speed up the process, we apply the odd cycle transversal algorithm separately to each connected component of G'. The value of k, i.e. the maximum number of nodes and edges to remove, is first set to 0 and then iteratively increased until a feasible solution is found.

[1]Note that the algorithm will never choose to discard both auxiliary nodes representing the same paired read link, since this would contradict the optimality of the algorithm.

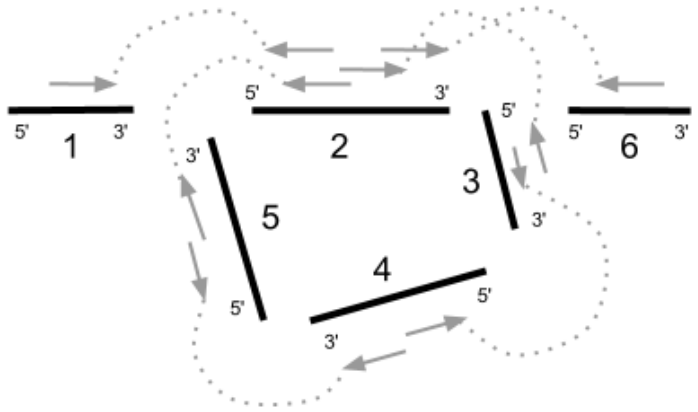

Figure 4.6: *Ordering problem.* Even though the given orientations of the contigs satisfy the paired read links, there is no consistent ordering of these contigs due to the cyclic nature of the links.

4.4 Ordering

At the end of the orientation stage, each contig is assigned a strand, providing us with a pairwise ordering of the contigs within each connected component. In the ordering step, we compute an absolute order of contigs for each component.

Since the contigs are now oriented, we represent the scaffolding problem as a directed graph T. Though the previous step ensures that there are no odd cycles left in G, there may still be cycles in T. Due to these cycles there might be no absolute order of the contigs that satisfies all pairwise constraints as illustrated in Figure 4.6.

To place the contigs into a linear order, we first need to eliminate all directed cycles from T. The problem of finding a minimal set of edges, whose removal makes a directed graph acyclic, is known as the feedback arc set problem. For arbitrary graphs, this problem is NP-hard [33]. We use a heuristic algorithm described in [21] which runs in $O(m)$ time and guarantees an upper bound of $m/2 - n/6$ where m is the number of edges and n is the number of nodes. As in the orientation stage, to improve accuracy and speed, we apply this algorithm to the connected components of T separately.

4.4.1 Spacing

During the ordering stage, T is transformed into a directed acyclic graph and is now guaranteed to have an ordering of the contigs so that the remaining links are satisfied. Yet, this ordering may not be unique. In the last stage of scaffolding, we try to find a placement of contigs within each scaffold such that the distances between the contigs agrees best with the size of the gaps as suggested by the paired read links. In a general context, this problem is formulated as a Mixed Integer Programming problem. In our case, we can instead formulate this problem as a Linear Programming (LP) problem since the orientations are already fixed.

In the LP formulation, for each contig $1 \leq i \leq N$, where N is the number of contigs in the scaffold, we have a real valued free variable x_i that represents the 5' coordinate of the contig. Without loss of generality, we set x_1 to 0. For each paired read link, we introduce the following constraints:

$$x_i - x_j + d_{ij} \leq C(1 - \delta_{ij}) \tag{4.3}$$

$$x_j - x_i - d_{ij} \leq C(1 - \delta_{ij}) \tag{4.4}$$

where d_{ij} is the distance between the 5' ends of the contigs i and j suggested by the paired read link. δ_{ij} is a real valued slack variable in the range $[0, 1]$. C is a large constant set to the sum of all the contig lengths and suggested distances between the contigs (or to the estimated genome length if the former exceeds the latter). Subject to the set of constraints as constructed above, we maximize $\sum_{i,j} \delta_{ij}$.

Although this formulation is designed to place the contigs so that the paired read links are satisfied best, it may allow two contigs to occupy the same coordinates. In practice, we do not use the coordinates returned by the LP solver; rather, we use these coordinates to order the contigs in linear paths. If a contig i significantly overlaps with

Table 4.1: *Datasets used for evaluation.* The accession codes for the *E. coli* and the *P. syringae* datasets are SRX000429 and ERX000536 respectively. Assemblathon1 dataset consists of artificial paired-end Illumina reads simulated with errors.

	Genome size (mbp)	No. of reads	Sequence Coverage	Read length (bp)	Insert size (bp)
E. coli	4.6	2×10.4m	160x	36	200
P. syringae	6.1	2×3.5m	40x	36	400
Assemblathon1	112.5	2×22.5m	40x	100	300

the next contig j and the length of the shortest path[2] between them in T is longer than a small threshold, we infer that they are not supposed to be adjacent. In this case, the contigs following j are considered in order until one of them passes these criteria. If such a contig is found, then it follows i in the path and a new path is created for j. The resulting linear paths are written as scaffolds.

4.5 Evaluation

In our first set of experiments, we compare ScaRPA to several other stand-alone scaffolders on two bacterial genomes: *E. coli* (strain K-12 substrain MG1655) and *P. syringae* (pathovar syringae B728a). For these genomes, we use Illumina paired-end libraries downloaded from the Short Read Archive (`http://www.ncbi.nlm.nih.gov/sra`). Next, we evaluate the combined performance of Hapsembler and ScaRPA against a state-of-the-art assembler that has a dedicated scaffolding module. In this experiment, we use an artificial diploid genome originally constructed for the first Assemblathon experiment [15]. This genome is simulated by computationally evolving a real genome and is available

[2]The path lengths here are calculated in terms of the number of edges.

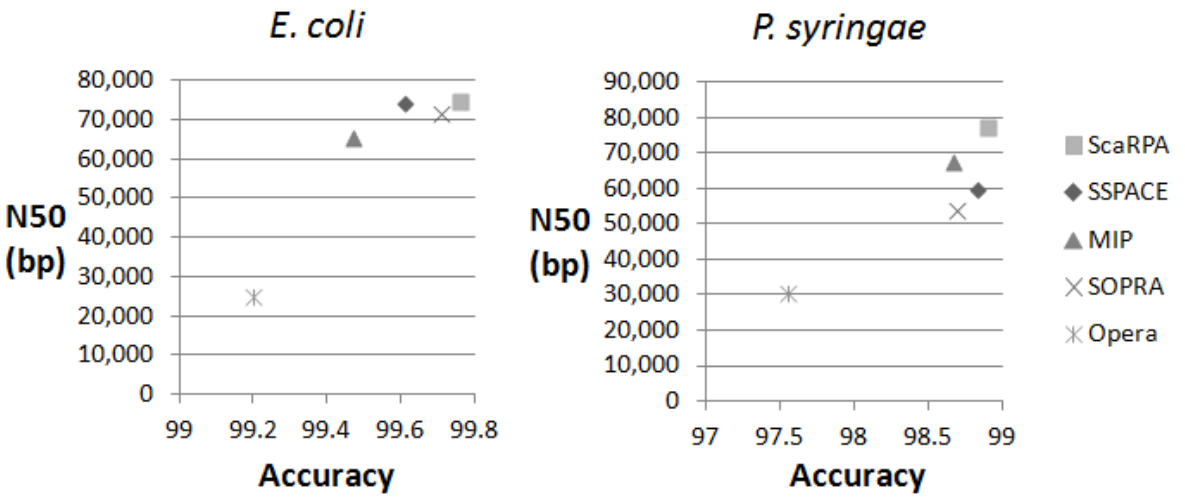

Figure 4.7: *Scaffold accuracy versus N50 for the E. coli and P. syringae datasets.*

as a diploid reference sequence. A SNP rate of about 1% is modeled in the simulation. Moderate levels of other types of polymorphism such as indels are also present. The statistics of all datasets are given in Table 4.1.

4.5.1 Scaffolding bacterial genomes

On the *E. coli* and *P. syringae* datasets, we compare ScaRPA to four other scaffolders; SSPACE [7], MIP Scaffolder, [63], SOPRA [13] and Opera [24]. We assemble the reads into contigs using Hapsembler (see Chapter 3) and only use contigs longer than 100bp for scaffolding. The standard deviation for each library is set to 10% of the mean insert size. For MIP Scaffolder, the minimum and maximum insert size values are set to 3 standard deviations below and above the mean respectively. When running Opera we set the PET threshold to the smallest value accepted by the program. The other parameters are left at default values. We let ScaRPA adjust all parameters automatically for both datasets. For all scaffolders, the reads are mapped with Bowtie [40].

The scaffolding results for these datasets are summarized in Tables 4.2 and 4.3. To evaluate the accuracy of the scaffolds, we employ a method similar to the one used in [63].

Table 4.2: *Scaffolding results for the E. coli dataset.* We define NG50 as the largest scaffold size such that the sum of scaffolds at least as long is greater than half the genome size. N50 is calculated using the total scaffold size reported by the scaffolder instead of the genome size. For each scaffolder, the second row contains the statistics calculated without gaps.

	No. of sequences	Accuracy at 3k (%)	Coverage (%)	Largest (bp)	N50 (bp)	NG50 (bp)	Total (mbp)
Contigs	371	100.00	98.08	82,795	23,195	22,156	4.57
ScaRPA	140	99.76	98.21	248,493	74,796	72,387	4.56
				248,461	74,790	72,311	4.56
SSPACE	142	99.61	98.17	178,229	74,044	74,044	4.56
				178,226	73,921	73,921	4.56
MIP	134	99.47	98.28	236,583	65,283	62,344	4.56
				236,450	65,283	62,302	4.56
SOPRA	171	99.71	98.10	180,052	71,485	67,663	4.57
				180,017	71,485	67,653	4.56
Opera	312	99.20	98.11	82,805	24,869	24,843	4.57
				82,795	24,859	24,833	4.57

Table 4.3: *Scaffolding results for the P. syringae dataset.* We define NG50 as the largest scaffold size such that the sum of scaffolds at least as long is greater than half the genome size. N50 is calculated using the total scaffold size reported by the scaffolder in place of the genome size. For each scaffolder, the second row contains the statistics calculated without gaps.

	No. of sequences	Accuracy at 3k (%)	Coverage (%)	Largest (bp)	N50 (bp)	NG50 (bp)	Total (mbp)
Contigs	876	99.70	98.17	51,433	12,947	12,866	6.03
ScaRPA	244	98.90	98.23	273,039	77,009	74,955	6.02
				272,420	76,699	74,804	6.01
SSPACE	237	98.83	98.20	227,759	59,665	59,665	6.03
				227,712	59,728	59,728	6.02
MIP	244	98.67	98.32	252,801	67,675	67,442	6.02
				252,586	67,333	67,299	6.01
SOPRA	387	98.69	98.19	262,140	53,635	53,635	6.03
				261,935	53,533	53,533	6.03
Opera	407	97.56	98.34	137,453	30,290	30,283	6.05
				137,274	30,263	30,250	6.03

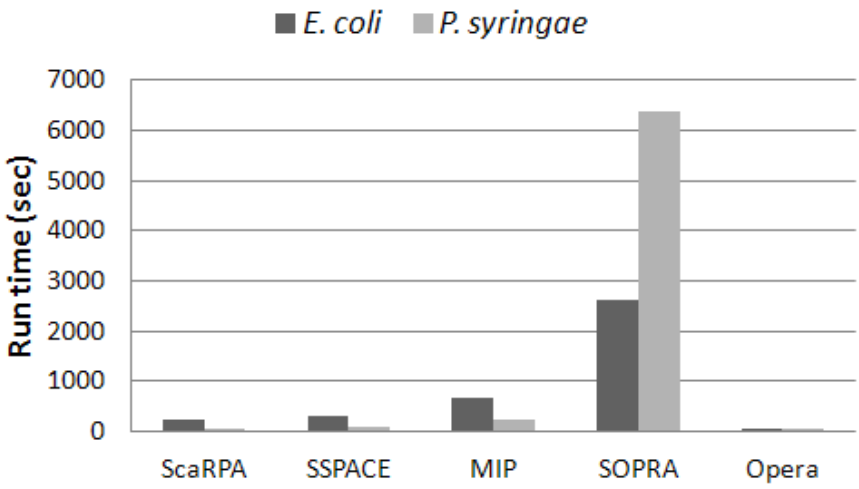

Figure 4.8: *Running times of the scaffolders for the E. coli and P. syringae datasets.* Mapping time is excluded for all scaffolders with the exception of SSPACE, which runs Bowtie internally. As a comparison, the total wall-clock times taken by Bowtie to index the reference and report read mappings are 288 seconds for *E. coli* and 94 seconds for *P. syringae* using 8 threads.

Briefly, this method works by extracting pairs of sequences separated by a certain distance from the scaffolds. These pairs are then mapped to the reference and the proportion of pairs that map with the correct orientation and within 10% of the correct distance is reported. In our experiments, we use 1000bp long sequences separated by a distance of 3000bp. We map the contigs and scaffolds to the reference sequences using MUMmer [38].

While the results are generally similar, ScaRPA produces the most accurate scaffolds for both datasets. For scaffold contiguity, we see that ScaRPA performs similarly to SSPACE, MIP Scaffolder and SOPRA, whereas Opera produces shorter scaffolds that are also less accurate (Figure 4.7). The running times of the scaffolders are given in 4.8. On these datasets, Opera is the fastest, followed by ScaRPA and SSPACE. SOPRA is the slowest among the five scaffolders taking over an hour on the *P. syringae* dataset.

4.5.2 Hapsembler + ScaRPA

On the Assemblathon1 dataset, we compare the combined performance of Hapsembler and ScaRPA to SOAPdenovo genome assembler [43], which includes a scaffolding module. As before, for SOAPdenovo we report the results using the k-mer size that gives the best N50 value. For validation, we use the same methods described above save that we map the contigs/scaffolds separately to each haploid reference and report the average for coverage and accuracy. The results are given in Table 4.4. Only contigs/scaffolds longer than 500bp are reported.

Hapsembler produces significantly longer contigs than SOAPdenovo, while the latter only covers ∼90% of the genome with contigs longer than 500bp. We remark that the higher accuracy of SOAPdenovo contigs is largely due to the lack of long contigs. Note that our validation method requires contigs of length ≥3000bp and SOAPdenovo produces much fewer contigs of this size compared to Hapsembler.

The scaffolds produced by ScaRPA using Hapsembler contigs are also substantially

Table 4.4: *Scaffolding results for the Assemblathon1 dataset.* We define NG50 as the largest scaffold size such that the sum of scaffolds at least as long is greater than half the genome size. N50 is calculated using the total scaffold size reported by the scaffolder in place of the genome size. The first section reports the contig statistics and the second section reports the scaffold statistics.

	No. of sequences	Accuracy at 3k (%)	Coverage (%)	Largest (bp)	N50 (bp)	NG50 (bp)	Total (mbp)
Hapsembler	8,592	99.72	97.89	130,810	22,901	22,917	112.60
SOAPdenovo	57,680	99.97	91.11	43,516	2,337	2,170	104.21
Hapsembler	5,816	99.37	97.92	245,922	38,665	38,665	112.53
+ ScaRPA				245,820	38,661	38,630	112.49
SOAPdenovo	29,217	96.18	95.39	60,848	9,252	9,315	113.24
				60,260	9,172	9,136	111.89

longer and more accurate than SOAPdenovo scaffolds. Moreover, even though SOAPdenovo reports a total of 113.24mbp of scaffolds, these scaffolds still cover less than 96% of the genome. In contrast, Hapsembler + ScaRPA scaffolds cover nearly 98% of the genome.

4.6 Discussion

The advantage of paired reads in genome assembly is two-fold: During the contig assembly stage, paired reads are useful in resolving repeat and polymorphism related issues; in the scaffolding stage, paired reads allow us to jump across coverage gaps and establish a relative order and orientation of the contigs. As we have seen in this chapter, clever utilization of paired reads during scaffolding can also help detect contig misassemblies.

Like contig assembly, scaffolding is a computationally challenging problem. The methods we present in this chapter achieve tractability using bounded parameters or near-optimal algorithms. For most scaffolders, the running time of scaffolding is typically dominated by the read mapping step on smaller genomes with high coverage depths. On the other hand, read mapping can be performed in parallel. As a result, for complex genomes, the scaffolding stage is expected to be the real bottleneck.

Within ScaRPA, the most time consuming step is the contig orientation task. While we believe our method produces more accurate scaffolds than greedy or heuristic based approaches, it can be computationally expensive for large and complex datasets. On the other hand, the fixed-parameter tractable algorithm we employ is suitable for parallel computation. Although our current implementation is single-threaded, we plan to explore this idea in a future version.

Chapter 5

Analysis of multiple substitution codons in *Ciona Savignyi*

5.1 Introduction

In the previous chapters, we have presented methods to facilitate the sequencing and assembly of highly polymorphic organisms. In this chapter, we present a case study of polymorphism in *C. savignyi* as a motivation for our efforts in developing these methods. As we explain below, sequencing of such organisms can be very useful in studying certain phenomena and patterns of evolution, which may be almost impossible to investigate in less polymorphic species.

Our case study considers one such phenomenon: the presence, in different haploid genotypes (haplotypes) of *C. savignyi*, of allelic codons that differ from each other at more than one nucleotide site. Such allelic codons are exceedingly rare in less diverse populations. When codons that differ from each other by multiple non-synonymous substitutions were observed in different species, an excess of substitutions within the same lineage was interpreted as a sign of positive selection [5, 4]. Here, using the closely related species *C. intestinalis* as an outgroup, we report a similar excess in two haplotypes

of *C. savignyi* and analyze its possible causes.

5.2 Material and methods

C. intestinalis annotations constituting 14,002 genes were downloaded from `ftp://ftp.jgi-psf.org/pub/JGI_data/Ciona/v2.0`. The *C. savignyi* genome (version 2.01) was downloaded from `http://mendel.stanford.edu/SidowLab/ciona.html`. We used the alignment of the two haploid genotypes, A and B, available at the site and described in [73].

5.2.1 Aligning gene triplets

Translated *C. intestinalis* genes were queried against both haplotypes of *C. savignyi*, using the protein2genome functionality of the alignment software Exonerate [71]. The best hits for a gene on both haplotypes were required to have a normalized score of at least 3.0 (calculated by dividing the Exonerate raw score by the protein length involved in the alignment). In case of a tie for the best hit in either of the haplotypes, or if the normalized score of the best hits on each haplotype differed by > 0.5, the protein was eliminated to avoid potential paralog problems. In addition, the locations of the alignments on each haplotype were checked to ensure that they correspond to the same position (± 5 nucleotides) in the global alignment of the two haplotypes [73]. If the intersection of the pairwise alignments was < 100 codons or covered $< 75\%$ of the gene, the gene was not considered for further analysis. Finally, alignments with ambiguities in nucleotide sequences or internal stop codons were discarded. For all of the remaining genes, the two sets of pairwise Exonerate alignments were merged into three-way alignments as follows: the intersection of *C. intestinalis* vs. haplotype A and *C. intestinalis* vs. haplotype B alignments was taken on the basis of the *C. intestinalis* amino acid, while introducing extra gaps whenever one of the pairwise alignments had a gap in the *C. intestinalis*

sequence that the other did not. The remaining data set consisted of 5478 triplets of orthologous genes.

We further masked codons that were not flanked, on each side, by gapless alignments of $\geq$ 10 amino acids, with at least five matches between the two haplotypes and at least three matches between each haplotype and *C. intestinalis*. To remove the effect of insertion and deletion sequencing errors, we also masked frame-shifted regions: those in which the same DNA sequence of length 4 or more occurred in the aligned *C. savignyi* sequences with a shift of ± 1.

Gene-specific synonymous and non-synonymous evolutionary distances were estimated by the Codeml program of the PAML package [78] from pairwise nucleotide alignments for the two *C. savignyi* haplotypes and each haplotype and the *C. intestinalis* genome, taken from the triple alignments. When the distances were estimated for correlation with the occurrence of a variable codon in some region, that codon itself was excluded in distance estimation.

5.2.2 Determination of the last common ancestor

The last common ancestor (LCA) codon for a pair of allelic codons in the two *C. savignyi* haplotypes was determined as follows. When the two allelic codons differed from each other at one nucleotide site, and encoded the same amino acid, we assumed that if the homologous *C. intestinalis* codon (outgroup, O) coincided either with the codon from haplotype A or with the codon from haplotype B, LCA coincided with O. In other words, we assumed parsimony, which implies that when O coincides with A (B), the single synonymous nucleotide substitution occurred in the lineage that led to B (A). If A and B differ from each other at one nucleotide site but encode different amino acids, we assumed that if O either coincides with A (B) or differs from both A and B but still encodes the same amino acid as A (B), the LCA also encodes this amino acid, and that the only non-synonymous substitution occurred in the lineage that led to B (A). When

O encodes an amino acid different from that encoded by both A and B, we assumed that the LCA could not be determined.

When the two allelic codons differ from each other at two nucleotide sites, we also assumed that the LCA coincided with O either if O coincided with A or B or if the O codon was intermediate between codons A and B, in the sense that O differed from both A and B by a single nucleotide. In addition, when both allelic codons and the two intermediates all encoded different amino acids, we assumed that the LCA encoded the same amino acid as O if O encoded the same amino acid as A, B, or one of the intermediate codons. Otherwise, we assumed that the LCA could not be determined. Pairs of codons for which one of the two possible intermediate codons is a stop codon were not considered.

When the two allelic codons differ from each other at three nucleotide sites, we assumed that the LCA coincided with O either if O coincided with A or B or if the O codon was intermediate between codons A and B, in the sense that O differed from one of them at one nucleotide site and from the other one at two nucleotide sites. In all other cases, we assumed that the LCA could not be determined, as it is usually impossible to establish with confidence that all the substitutions between the two allelic codons were non-synonymous. Pairs of codons for which one of the six possible intermediate codons is a stop codon were not considered.

5.3 Results

Among the 1,251,343 homologous codons in the 5478 analyzed genes, 93.46% are identical in the two haplotypes, and 6.40, 0.12, and 0.005% differ at one, two, and three nucleotide sites, respectively (no-, one-, two- and three-substitution codons). The mean evolutionary distance between the haplotypes is 0.086 at synonymous sites and 0.004 at non-synonymous sites, in agreement with [72]. Among codons with a single synonymous

Table 5.1: *Divergence at codons where haplotypes A and B differ at one nucleotide site.* Percentages for lineage A and lineage B are given for the codons where the last common ancestor (LCA) is known.

	Substitution in lineage of A (%)	Substitution in lineage of B (%)	LCA unknown (%)
synonymous	20,023 (50.0)	20,052 (50.0)	31,901 (44.3)
non-synonymous	2,452 (49.7)	2,481 (50.3)	3,236 (39.6)

Table 5.2: *Divergence at codons where haplotypes A and B differ at two nucleotide sites.* Percentages for the first and second columns are for codons where the last common ancestor (LCA) is known.

	Both substitutions in same lineage (%)	Substitutions in different lineages (%)	LCA unknown (%)
2 synonymous	82 (43.9)	105 (56.2)	101 (35.1)
1 synonymous and 1 non-synonymous	117 (43.0)	155 (57.0)	404 (59.8)
2 synonymous or 2 non-synonymous	52 (69.3)	23 (30.7)	69 (47.9)
2 non-synonymous	101 (65.6)	53 (34.4)	95 (38.2)

Table 5.3: *Distribution of codons where haplotypes A and B differ with 2 non-synonymous substitutions.*

	Both substitutions in same lineage (%)	Substitutions in different lineages (%)	LCA unknown (%)
Codons			
Possible false excess	63 (68.5)	29 (31.5)	74 (44.6)
Possible false deficit	38 (61.3)	24 (38.7)	21 (25.3)
1,3-substitution[a]	13 (56.5)	10 (43.5)	10 (30.3)
CpG-free[b]	69 (66.3)	35 (33.7)	62 (37.3)
Regions[c]			
Very strong conservation	12 (92.3)	1 (7.7)	4 (23.5)
Strong conservation	15 (78.9)	4 (21.1)	11 (36.7)
Moderate conservation	19 (55.9)	15 (44.1)	18 (34.6)
All others	55 (62.5)	33 (37.5)	62 (41.3)
Genes[d]			
Low D_n	18 (94.7)	1 (5.3)	4 (17.4)
Medium D_n	28 (58.3)	20 (41.7)	32 (40.0)
High D_n	55 (63.2)	32 (36.8)	59 (40.4)

a Codons where the two *C. savignyi* haplotypes differ from each other at the first and third nucleotide sites.

b Codons in which neither of the two possible intermediate states between haplotype A and haplotype B codons includes CpG context, either inside the codon or on its boundary.

c Regions with very strong, strong, and moderate conservation are those in which the codon under consideration is flanked from each side by gapless alignments of two *C. savignyi* genomes and C. intestinalis of length $\geq$10 each with 9 or 10, 8, and 7 invariant amino acids, respectively.

d Genes were split into three bins of equal size (low, medium, and high D_n) according to the average of D_n values between C. intestinalis and each of the haplotypes of *C. savignyi*.

substitution between haplotype A and haplotype B, O coincides with either A or B in 56% of the cases (Table 5.1). Among codons with a single non-synonymous substitution between A and B, O encoded the same amino acid as either A or B in 60% of the cases (Table 5.1).

There are 1610 codons separated by two substitutions, including 288 codons separated by two synonymous substitutions (such codons are rare, as they must code for either Leu or Arg) and 249 codons separated by two non-synonymous substitutions (Table 5.2). If the substitutions were independent, we would expect both of them to occur in the same lineage with the probability of ∼50% [5]. In agreement with this expectation, in codons where both substitutions were synonymous, they occur in the same lineage in approximately half of the cases. In contrast, two non-synonymous substitutions occur in the same lineage in 66% of the cases and in different lineages in only 34% of the cases. Clumping of non-synonymous substitutions is more pronounced in highly conserved genes and gene regions (Table 5.3). When two non-synonymous substitutions occur in the same lineage, they tend to occur in the lineage that has a higher rate of non-synonymous (86 of 101; chi square, $P < 0.0001$), but not necessarily synonymous (55 of 101; chi square, $P = 0.573$), substitutions in this gene. In contrast, when two synonymous substitutions occur in the same lineage, there is no significant difference in the rate of synonymous or non-synonymous substitutions between the two lineages (chi square, $P > 0.05$).

5.4 Analysis

In the following sections, we consider three possible explanations for the observed clumping of non-synonymous substitutions: positive selection, compensatory mutations, and potential biases in the last common ancestor identification, as well as other biases and phenomena that could lead to an excess of substitutions in the same lineage.

5.4.1 Two-substitution polymorphisms due to positive selection

Positive selection can lead to clumping of non-synonymous substitutions within a codon [5, 4]. However, in contrast to the previous observations of such clumping, here we are dealing with intrapopulation polymorphisms, and transitive polymorphisms due to positive selection-driven allele replacements are short-lived (see [12]). Thus, it is not clear whether positive selection driving both of the substitutions that convert the codon found in haplotype A into the codon found in haplotype B can provide a quantitatively feasible explanation. We roughly estimate the number of codons that differ by two non-synonymous substitutions between two haplotypes that segregate within a population under positive selection. Let us assume (unrealistically) that both of these substitutions occur simultaneously. Then, in the absence of dominance, the deterministic dynamics of replacement of codon A with codon B are described by the fundamental equation:

$$dp/dt = sp(1-p) \tag{5.1}$$

where p is the frequency of codon B, $1-p$ is the frequency of codon A, and s is the selective advantage of codon B over codon A. The solution to this equation is given by:

$$p(t) = \frac{1}{1+(1/p_0-1)e^{-st}} \tag{5.2}$$

(see [12]). If two haploid genotypes are sampled from the population every generation, the total number of heterozygous combinations over the whole course of a substitution is

$$T = \int_{\infty}^{\infty} 2p(t)(1-p(t))dt = 2/s \tag{5.3}$$

Thus, if we observe k heterozygous loci under positive selection within a pair of complete haploid genotypes, the per-generation number of positive selection-driven allele replacements required to explain this fact is $ks/2$. The excess of codons where both non-synonymous substitutions occurred in the lineage of the same haplotype sug-

gests that $k \sim 50$ (Table 5.2). Thus, even if selection is very weak (say, $s = 10^{-5}$), we still need to assume that there is a positive selection-driven replacement that results in a two-substitution codon every 4000 generations. Probably this rate of positive selection-driven evolution is too high [23], because selection that simultaneously favors two non-synonymous substitutions must occur only in a minority of cases. Moreover, our calculations underestimate the required prevalence of positive selection, because in reality the two substitutions necessary to convert codon A into codon B cannot occur simultaneously. Codon A will be replaced not by codon B directly, but by an intermediate codon that has selective advantage over A, and codons A and B would coexist for a much shorter period than the two alleles where one directly replaces the other, as assumed in Equation 5.1. Thus, simple positive selection favoring both changes is unlikely to be the leading cause of the observed pattern.

5.4.2 Two-substitution polymorphisms due to compensatory mutations

A second potential explanation for the observed pattern is compensatory evolution. Let us assume that codons A and B have the same fitness, but the intermediate codons have a reduced fitness, $1 - s$. In this case, in each pair of the substitutions, only the second (from an intermediate codon to either A or B) is driven by positive selection, and there is no long-term increase in fitness. Then, the deterministic equilibrium frequency of the intermediate codons will be $2m/s$, where m is the per-nucleotide mutation rate, and codons A and B appear from these intermediate codons, due to mutation, at the rate $m * (2m/s) = 2m^2/s$. If we treat coexistence of A and B as a selectively neutral polymorphism with the effective mutation rate" $m_{eff} = 2m^2/s$, the expected nucleotide diversity is $\pi = 4N_e m_{eff} = 8Nem^2/s$. Assuming $m = 10^{-8}$, $N_e = 10^6$ [72], and $s = 10^{-5}$, we obtain $\pi \sim 10^{-4}$. Thus, to explain the 50 extra codons where the variants found in haplotypes A and B differ by two non-synonymous substitutions (Table 5.2), we need to

assume that such compensatory selection operates on 5×10^5 codons, i.e., on 40% of all codons. The assumption that such a large fraction of protein sites are under this kind of selection, with at least two amino acids conferring the same high fitness, does not seem to be very likely. Of course, if the selection against the intermediate codons is weaker, a smaller fraction of codons under compensatory selection will suffice: if $s = 10^{-6}$, only 4% of the codons could be under this selection. As s declines past 10^{-6}, however, selection becomes inefficient given $N_e = 10^6$, and the intermediate codon(s) will become effectively neutral.

5.4.3 Biased misidentification of the ancestral codon

The observed clumping could also be caused by biased misidentification of the LCA codon for the two *C. savignyi* haplotypes. Because *C. intestinalis* is not a close outgroup for within-species polymorphism in *C. savignyi*, in a substantial fraction of cases, the LCA cannot be identified under the assumption of parsimony, and, in some cases, the LCA is likely to be misidentified. Unbiased mistakes in identification of the LCA codon will not produce the observed pattern: if, in the case of a two-substitution pair of *C. savignyi* codons A and B, the LCA codon was drawn randomly from the four possible codons (A, B, and the two intermediates), the two substitutions that distinguish A from B will be attributed to the same and the two different haplotypes with equal probabilities. However, there may also be a systematic bias in misidentification of LCA, due to two reasons.

First, it may be possible that only one of the two intermediate codons confers a high fitness and the other one confers a low fitness. For example, if the codons A and B are AAT (encoding Asn) and GGT (encoding Gly), the intermediate codon AGT (encoding Ser) may be fit, and the intermediate codon GAT (encoding Asp) may be unfit. Then, if the outgroup is very distant from the LCA, it will carry the fit intermediate codon in only one-third of cases (assuming that at equilibrium codons AAT, GGT, and AGT are

equally common). As a result, one could conclude that for two-thirds of codon pairs, both substitutions occurred in one *C. savignyi* haplotype, because the LCA (as revealed by the outgroup) coincides with the other haplotype.

Unfortunately, *C. intestinalis* is the closest known outgroup that can be used to polarize the *C. savignyi* polymorphism. We investigated the potential effect of biased misidentification of the LCA by testing the robustness of the excess of multiple substitutions in the same lineage in interspecific divergence [5] to the choice of the outgroup. We identified the codons with two non-synonymous substitutions between human and mouse and determined the LCA in three ways: (i) using only sites where dog and opossum carry the same codon (most confident), (ii) using dog as an outgroup, and (iii) using opossum as an outgroup (least confident). The corresponding fractions of codons where both substitutions in two-substitution human-mouse codons occurred in the same lineage (amino acid-level pattern) were 76, 69, and 67%, respectively. Thus, the excess of codons where both substitutions were attributed to the same lineage diminishes when a more distant outgroup is used. These data argue against biased misidentification of the LCA as an explanation of the observed pattern in divergence between independent evolutionary lineages, although this conclusion does not necessarily apply to within-species polymorphism of *C. savignyi*.

Second, biased misidentification of the LCA may appear if the amino acid composition of proteins is out of equilibrium [31]. If, for a double-substitution codon, the amino acid substitution that creates the terminal amino acid from the intermediate amino acid is more likely than the reciprocal substitution, some clumping could be an artifact of systematic errors in inferring the LCA from the outgroup codon [5]. To test this hypothesis, we used the data on codons identical between the two *C. savignyi* haplotypes to infer the rates of each non-synonymous substitution between *C. intestinalis* and *C. savignyi*. Next, for each two-substitution pair of codons, we compared the rate of substitutions from each of the two intermediate codons into each of the two terminal codons (a total

of four substitutions) with the rate of the reciprocal substitutions. A substitution from intermediate to terminal codon can lead to false excess of clumping for the given two-substitution codon if its rate is higher than the rate of the reciprocal substitution. We compared the clumping between two-substitution codons with zero or one false excess amino acid substitutions (false deficit codons) with that in two-substitution codons with two to four false excess amino acid substitutions (false excess codons). The difference between the two values was not large (chi square, $P > 0.1$), arguing against nonequilibrium composition of proteins as a leading cause for the observed clumping.

5.4.4 Other explanations

The observed clumping might also be due to errors in alignment or closely correlated nearby single-nucleotide sequencing errors. We believe, however, that our filtering criteria are effective in eliminating such cases: we make sure that the two orthologous *C. savignyi* exons are also aligned to each other in the haplotype alignments and that the level of conservation between the two exons in *C. savignyi* is above a threshold. The observed elevated prevalence of non-synonymous substitutions in the vicinity of codons where haplotypes A and B differ by two non-synonymous substitutions is unlikely to be due to sequencing errors, because this effect is absent when A and B differ by two synonymous substitutions. Moreover, the clumping of non-synonymous substitutions is the strongest when the nearby codons are completely conserved between the A and B haplotypes, further reducing the chance that sequencing or alignment errors play a dominant role.

This clumping cannot be explained by hypermutable CpG dinucleotides, because substitutions are also clumped in codon pairs where no intermediate codons contain CpGs (Table 5.3). Due to the small number of two-substitution amino acids with mutations at the first and third nucleotides, we cannot immediately dismiss that some of the clumping is due to mutation events spanning two adjacent nucleotides. In the interspecific case, however, [5] rejected this explanation in the presence of a larger data set.

5.5 Discussion

Our analysis of differences between the two haploid genotypes from one *C. savignyi* individual follows closely that for differences between mouse and rat genomes [5]. Surprisingly, the results of these analyses are also rather similar to each other. Our data reveal clumping of within-population non-synonymous polymorphisms at the same codon that is similar in magnitude to the clumping of non-synonymous substitutions that distinguish different genomes. Clumping of non-synonymous substitutions in different mammalian species [5] and HIV-1 strains [4] was interpreted as the signature of positive selection that occurred at some time during the divergence of the analyzed lineages. Although the exact fraction of positive selection-driven amino acid substitutions remains controversial, it is almost certainly substantial [23].

In contrast, the role of positive selection in maintaining polymorphism at the molecular level is believed to be small [36]. Indeed, our rough estimates suggest that the observed clumping of non-synonymous substitutions cannot be easily explained by positive selection. We considered two different scenarios, assuming that either both or only one of the two non-synonymous differences between the two *C. savignyi* codons are favored by positive selection, and in both cases it takes very high prevalence of positive selection to explain the observed clumping. Another feasible explanation for the observed clumping of non-synonymous substitutions is the biased mis-identification of the LCA. Qualitatively, this is feasible if only one intermediate codon has a high enough fitness to be present. Quantitatively, however, this mechanism appears to be unlikely to explain what we see. We also considered the effect of sequencing errors and correlated mutations at adjacent nucleotides and do not believe these to be the leading causes of the observed clumping.

Recently, Schrider *et al.* [66] has suggested that the clumping of non-synonymous substitutions may be due to multinucleotide mutational events. In humans, this explanation is particularly appealing because a similar clumping is observed even in parent-offspring

trios [66]. Nevertheless, the available data is not adequate to draw the same conclusion for *C. savignyi*. In particular, the allele frequencies of intermediate codons will help elucidate the factors responsible for these observations. Deriving these frequencies requires the sequencing and analysis of additional haplotypes. Automated methods that can handle extreme polymorphism, as described in this thesis, can greatly facilitate this task.

Chapter 6

Concluding Remarks

Like many interesting problems in computational biology, genome assembly is a computationally hard problem. Despite this computational difficulty, the human genome - complex and monumental in size - was assembled more than a decade ago using only a fraction of the computational power available to us today. With the new challenges and opportunities introduced by the recent innovations in sequencing technologies, genome assembly is likely to remain as an active research field for many years to come.

Written in a time when the sequencing technologies and their applications are rapidly evolving, this thesis exhibits several interesting phenomena and trends.

In Chapter 2, we demonstrate that quality scores, when incorporated in a robust probabilistic framework, can be very useful in distinguishing errors from polymorphisms. Yet a majority of error correction tools and genome assemblers, developed for the new sequencing platforms, ignore quality scores. This trend is in contrast with the tools developed for Sanger sequencing, for which the quality scores played a prominent role. In most cases, the underlying reason for this is the memory overhead. We believe that this is an omissible problem which can be mitigated by clever software implementations and hope that our results may inspire future research in this direction.

In genome assembly, a noticeable trend is the exclusive use of the de Bruijn Graph

approach for the new sequencing platforms. As we have discussed in Chapter 3, this approach is typically sensitive to sequencing errors. In particular, insertion and deletion errors, which are common in the Roche/454 platform, can not be resolved via k-mer correction methods such as Spectral Alignment.

Another drawback of this approach is the loss of information due to the breaking of reads into k-mers. For very short reads, as produced by the earlier versions of the existing high throughput platforms, this loss of information is insignificant. As the read lengths of these platforms rapidly increase, the utility of the de Bruijn graphs may be questionable.

In contrast, the Overlap-Layout-Consensus approach can utilize longer reads more effectively. Combined with advanced algorithms that exploit paired sequencing, in Chapters 3 and 4, we have demonstrated that this approach can outperform state-of-the-art de Bruijn Graph based assemblers on relatively large diploid genomes. On the other hand, the Overlap-Layout-Consensus approach, in its basic form, is arguably inefficient to assemble very large genomes with the current read lengths and coverage depths produced by the high throughput sequencers.

We speculate that the main challenge of genome assembly in the near future will be to efficiently assemble complex genomes to a high degree of accuracy with nearly Sanger-length reads and very high coverage. We hope that the methods described in this thesis will provide insights to help achieve this goal.

Bibliography

[1] Genome 10k: A proposal to obtain whole-genome sequence for 10000 vertebrate species. *Journal of Heredity*, 100(6):659–674, 2009.

[2] S. F. Altschul and D. J. Lipman. Trees, stars, and multiple biological sequence alignment. *SIAM Journal on Applied Mathematics*, 49(1):197–209, 1989.

[3] S. Batzoglou, D. B. Jaffe, K. Stanley, and J. Butler. Arachne: a whole-genome shotgun assembler. *Genome Research*, pages 12–177, 2002.

[4] G. A. Bazykin, J. Dushoff, S. A. Levin, and A. S. Kondrashov. Bursts of non-synonymous substitutions in HIV-1 evolution reveal instances of positive selection at conservative protein sites. *Proceedings of the National Academy of Sciences*, 103:19396–19401, 2006.

[5] G. A. Bazykin, F. A. Kondrashov, A. Y. Ogurtsov, S. Sunyaev, and A. S. Kondrashov. Positive selection at sites of multiple amino acid replacements since rat-mouse divergence. *Nature*, 429:558–562, 2004.

[6] D. Bentley *et al*. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456:53–59, 2008.

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

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

[9] M. J. Chaisson, D. Brinza, and P. A. Pevzner. *De novo* fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research*, 19(2):336–346, 2009.

[10] B. Chevreux, T. Pfisterer, B. Drescher, A. J. Driesel, W. E. Muller, T. Wetter, and S. Suhai. Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs. *Genome Research*, 14(6):1147–1159, 2004.

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

[12] J. F. Crow and M. Kimura. *An introduction to population genetics theory.* Harper & Row, New York, 1970.

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

[14] P. Dehal *et al.* The draft genome of *Ciona intestinalis*: Insights into chordate and vertebrate origins. *Science*, 298(5601):2157–2167, 2002.

[15] A. E. Dent *et al.* Assemblathon 1: A competitive assessment of *de novo* short read assembly methods. *Genome Research*, 21(12):2224–2241, 2011.

[16] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[17] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. SHARCGS: a fast and highly accurate short-read assembly algorithm for *de novo* genomic sequencing. *Genome Research*, 17(11):1697–1706, 2007.

[18] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[19] N. Donmez, G. Bazykin, M. Brudno, and A. S. Kondrashov. Polymorphism due to multiple amino acid substitutions at a codon site within *Ciona savignyi*. *Genetics*, 181:685–690, 2009.

[20] N. Donmez and M. Brudno. Hapsembler: an assembler for highly polymorphic genomes. In V. Bafna and S. Sahinalp, editors, *Research in Computational Molecular Biology*, volume 6577 of *Lecture Notes in Computer Science*, pages 38–52. Springer Berlin / Heidelberg, 2011.

[21] P. Eades, X. Lin, and W. F. Smyth. A fast effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47:319–323, 1993.

[22] B. Ewing, L. Hillier, M. C. Wendl, and P. Green. Base-calling of automated sequencer traces using Phred error probabilities. *Genome Research*, 8:175–185, 1998.

[23] A. Eyre-Walker. The genomic rate of adaptive evolution. *Trends in Ecology & Evolution*, 21(10):569–575, 2006.

[24] S. Gao, N. Nagarajan, and W.-K. Sung. Opera: Reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. In V. Bafna and S. Sahinalp, editors, *Research in Computational Molecular Biology*, volume 6577 of *Lecture Notes in Computer Science*, pages 437–451. Springer Berlin / Heidelberg, 2011.

[25] D. Hernandez, P. Francois, L. Farinelli, M. Osteras, and J. Schrenzel. *De novo* bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*, 18(5):802–809, 2008.

[26] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9(9):868–877, 1999.

[27] D. H. Huson, K. Reinert, and E. W. Myers. The greedy path merging algorithm for contig scaffolding. *Journal of the ACM*, 49(5):6003–615, 2002.

[28] R. M. Idury and M. S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2:291–306, 1995.

[29] L. Ilie, F. Fazayeli, and S. Ilie. HiTEC: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302, 2011.

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

[31] I. K. Jordan, F. A. Kondrashov, I. A. Adzhubei, Y. I. Wolf, and E. V. Koonin *et al*. A universal trend of amino acid gain and loss in protein evolution. *Nature*, 433:633–638, 2005.

[32] W.-C. Kao, A. H. Chan, and Y. S. Song. ECHO: A reference-free short-read error correction algorithm. *Genome Research*, 21(7):1181–1192, 2011.

[33] R. M. Karp. Reducibility among combinatorial problems. *Complexity of computer computations*, pages 85–103, 1972.

[34] J. Kececioglu. *Exact and Approximation Algorithms for DNA Sequence Reconstruction*. PhD dissertation, Department of Computer Science, University of Arizona, 1991.

[35] D. Kelley, M. Schatz, and S. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11:R116, 2010.

[36] M. Kimura. *The neutral theory of molecular evolution*. Cambridge University Press, Cambridge, UK, 1983.

[37] M. Kircher and J. Kelso. High-throughput DNA sequencing concepts and limitations. *BioEssays*, 32(6):524–536, 2010.

[38] S. Kurtz, A. Phillippy, A. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[39] T. Kvist, B. Ahring, R. Lasken, and P. Westermann. Specific single-cell isolation and genomic amplification of uncultured microorganisms. *Applied Microbiology and Biotechnology*, 74:926–935, 2007.

[40] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3), 2009.

[41] J. Laserson, V. Jojic, and D. Koller. Genovo: *de novo* assembly for metagenomes. In B. Berger, editor, *Research in Computational Molecular Biology*, volume 6044 of *Lecture Notes in Computer Science*, pages 341–356. Springer Berlin / Heidelberg, 2010.

[42] J. Li, H. Jiang, and W. Wong. Modeling non-uniformity in short-read rates in RNA-Seq data. *Genome Biology*, 11(5), 2010.

[43] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang. *De novo* assembly of human genomes

with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, 2009.

[44] D. Lokshtanov, S. Saurabh, and S. Sikdar. Simpler parameterized algorithm for OCT. In J. Fiala, J. Kratochvíl, and M. Miller, editors, *Combinatorial Algorithms*, pages 380–384. Springer Berlin / Heidelberg, 2009.

[45] E. R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends in Genetics*, 24(3):133 – 141, 2008.

[46] A. Masella, A. Bartram, J. Truszkowski, D. Brown, and J. Neufeld. PANDAseq: paired-end assembler for illumina sequences. *BMC Bioinformatics*, 13(1):31, 2012.

[47] P. Medvedev, K. Georgiou, E. W. Myers, and M. Brudno. Computability of models for sequence assembly. In *Proceedings of WABI*, pages 289–301, 2007.

[48] P. Medvedev, S. Pham, M. Chaisson, G. Tesler, and P. Pevzner. Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. In V. Bafna and S. Sahinalp, editors, *Research in Computational Molecular Biology*, volume 6577 of *Lecture Notes in Computer Science*, pages 238–251. Springer Berlin / Heidelberg, 2011.

[49] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner. Error correction of high-throughput sequencing datasets with non-uniform coverage. *Bioinformatics*, 27(13):137–141, 2011.

[50] B. Mishra. Statistical distributions in genome sequence assembly. www.cs.nyu.edu/faculty/mishra/COURSES/09.HPGP/Lecture4BioX, Lecture Notes, last accessed date 7/8/2012.

[51] E. W. Myers. The fragment assembly string graph. *Bioinformatics*, 21:79–85, 2005.

[52] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. J. Reinert, K. A. Remington, E. L. Anson, A. A. Bolanos, Q. Zhang, X. Zheng, G. M. Rubin, M. D. Adams, and J. C. Venter. A whole-genome assembly of drosophila. *Genome Research*, 287(5461):2196–2204, 2000.

[53] N. Nagarajan, T. D. Read, and M. Pop. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, 24(10):1229–1235, 2008.

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

[55] P. A. Pevzner, H. Tang, and M. S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[56] M. Pop, D. Kosack, and S. Salzberg. Hierarchical scaffolding with Bambus. *Genome Research*, 14:149–159, 2004.

[57] M. Pop and S. L. Salzberg. Bioinformatics challenges of new sequencing technology. *Trends in Genetics*, 24(3):142 – 149, 2008.

[58] K. Rasmussen, J. Stoye, and E. Myers. Efficient q-gram filters for finding all e-matches over a given length. *Journal of Computational Biology*, 13:296–308, 2005.

[59] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32:299–301, 2004.

[60] M. Ronaghi, S. Karamohamed, B. Pettersson, M. Uhlen, and P. Nyren. Real-time DNA sequencing using detection of pyrophosphate release. *Analytical Biochemistry*, 242(1):84–89, 1996.

[61] S. M. Rumble, P. Lacroute, A. V. Dalca, M. Fiume, A. Sidow, and M. Brudno. Shrimp: Accurate mapping of short color-space reads. *PLoS Computational Biology*, 5(5), 05 2009.

[62] L. Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26:1284–1290, 2010.

[63] L. Salmela, V. Makinen, N. Valimaki, J. Ylinen, and E. Ukkonen. Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, 27(23):3259–3265, 2011.

[64] L. Salmela and J. Schroder. Correcting errors in short reads by multiple alignments. *Bioinformatics*, 27(11):1455–1461, 2011.

[65] F. Sanger and A. R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology*, 94(3):441–446, 1975.

[66] D. R. Schrider, J. N. Hourmozdi, and M. W. Hahn. Pervasive multinucleotide mutational events in eukaryotes. *Current Biology*, 21(12):1051–1054, 2011.

[67] J. Schroder, H. Schroder, S. J. Puglisi, R. Sinha, and B. Schmidt. SHREC: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.

[68] J. Shendure, G. J. Porreca, N. B. Reppas, X. Lin, J. P. McCutcheon, A. M. Rosenbaum, M. D. Wang, K. Zhang, R. D. Mitra, and G. M. Church. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, 309(5741):1728–1732, 2005.

[69] J. T. Simpson and R. Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373, 2010.

[70] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[71] G. S. Slater and E. Birney. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, 6(31), 2005.

[72] K. S. Small, M. Brudno, M. M. Hill, and A. Sidow. Extreme genomic variation in a natural population. *Proceedings of the National Academy of Sciences*, 104(13):5698–5703, 2007.

[73] K. S. Small, M. Brudno, M. M. Hill, and A. Sidow. A haplome alignment and reference sequence of the highly polymorphic *Ciona savignyi* genome. *Genome Biology*, 8(3), 2007.

[74] E. Sodergren *et al*. The genome of the sea urchin *Strongylocentrotus purpuratus*. *Science*, 314:941–952, 2006.

[75] G. G. Sutton, O. White, M. D. Adams, and A. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science & Technology*, 1:9–19, 1995.

[76] J. P. Vinson, D. B. Jaffe, K. O'Neill, E. K. Karlsson, N. Stange-Thomann, S. Anderson, J. P. Mesirov, N. Satoh, Y. Satou, C. Nusbaum, B. Birren, J. E. Galagan, and E. S. Lander. Assembly of polymorphic genomes: Algorithms and application to *Ciona savignyi*. *Genome Research*, 15(8):1127–1135, 2005.

[77] R. L. Warren, G. G. Sutton, S. J. M. Jones, and R. A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500–501, 2007.

[78] Z. Yang. PAML: a program package for phylogenetic analysis by maximum likelihood. *Computer Applications in the Biosciences*, 13:555–556, 1997.

[79] D. R. Zerbino and E. Birney. Velvet: Algorithms for *de novo* short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

[80] D. R. Zerbino, G. K. McEwen, E. H. Margulies, and E. Birney. Pebble and Rock Band: Heuristic resolution of repeats and scaffolding in the Velvet short-read *de novo* assembler. *PLoS ONE*, 4(12), 12 2009.