

## ABSTRACT

Title of Document:

DETECTING AND CORRECTING  
ERRORS IN GENOME ASSEMBLIES

Poorani Subramanian, Ph.D. 2010

Directed by:

Professor James A. Yorke, Department of  
Mathematics

Genome assemblies have various types of deficiencies or *misassemblies*. This work is aimed at detecting and correcting a type of misassembly that we call *Compression/Expansion* or *CE* misassemblies whereby a section of sequence has been erroneously omitted or inserted in the assembly. Other types of deficiencies include gaps in the genome sequence.

We developed a statistic for identifying Compression/Expansion misassemblies called the *CE statistic*. It is based on examining the placement of mate pairs of reads in the assembly. In addition to this, we developed an algorithm that is aimed at closing gaps and validating and/or correcting CE misassemblies detected by the CE statistic. This algorithm is similar to a shooting algorithm used in solving two-point boundary value problems in partial differential equations. We call this algorithm the Shooting Method. The Shooting Method finds all possible ways to assemble a local region of the genome contained between two target reads.

We use a combination of the CE statistic and Shooting Method to detect and correct some CE misassemblies and close gaps in genome assemblies. We tested our techniques both on faux and real data. Applying this technique to 22 bacterial draft assemblies for which the finished genome sequence is known, we were able to identify 5 out of 8 real CE misassemblies. We applied the Shooting Method to a de novo assembly

of the *Bos taurus* genome made from Sanger data. We were able to close 9,863 gaps out of 58,386. This added 8.34 Mbp of sequence to the assembly, and resulted in a 7 % increase of N50 contig size.

# **Detecting and Correcting Errors in Genome Assemblies**

by

Poorani Subramanian

Dissertation submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
2010

Advisory Committee:  
Professor James Yorke, Chair/Advisor  
Professor Brian Hunt  
Professor Mei-Ling Ting Lee  
Professor Konstantina Trivisa  
Dr. Aleksey Zimin

© Copyright by  
Poorani Subramanian  
2010

*for my parents, and my sister*

## Acknowledgments

Graduate school is a trying experience. As a graduate student, I've never been paid enough from my stipend to afford a television. But recently, a friend of mine (a grown-up with a job) asked me to housesit. His television is enormous, and all manner of wonderful programs are available by sifting through a menu and pushing a button. It was in this way that I discovered the joys of watching hours upon hours of reality television. People complain about reality television. They say it is manufactured to put the contestants under undue pressure. The challenges are difficult to complete in the allotted time. The judging is capricious. They say there is nothing real about it. But, I beg to differ. It may not reflect the reality most people face in life, but for a small number of individuals, this is their lives writ large on a 50 inch screen. These few are Ph.D. students.

To this end, I must thank a number of friends and family and strangers who helped me make it to the final challenge, who never once attempted to vote me off (although I'm sure they wanted to many times), and who texted my name in week after week.

**First, my parents.** When I told them I was saying no to my 75% scholarship to a top 25 law school my senior year of college to pursue a degree in mathematics, and that it would entail an extra year of undergrad work, my mother with years of immigrant guilt behind her said, "we didn't come to this country and work hard from nothing so you could throw your life away." That was their only lapse in support. Since then, they have been unflinching even with all my second-guessing, and they are the biggest reason this dissertation got written.

**My sister.** My sister is the super successful one in our family. She's an enormously busy trauma surgeon, but whatever small accomplishment I achieved along the way (even if it was just passing an exam), she always took the time to send me something special.

**The Shepard kids.** I couldn't figure out the social scene in graduate school, and I would sit at home on a Friday night and wonder why no one liked me. There was always an NU friend who I could chat with who would cheer me up with a, "your humor is just too cerebral for those kids," even when that was patently untrue. I'm just not that funny. Especially, Lev Guter, the best friend a girl could have, Kate MacLean, Tanya DeGroot, Heather Ballard, Ryan Zimmerman, and Claire (O'Connor) Battle.

**My fellow graduate students.** *We few, we happy few, we band of brothers; For he to-day that sheds his blood with me Shall be my brother;* (Shakespeare, Henry V). Carter Price, Fernando Galaz-Garcia, Guillaume Marçais and Amy Finkbiner.

**My friends in the DMV.** The people who copy-edited the final draft of my dissertation, bought my drinks, tolerated my fickle tendency to cancel because of "work" (which was really just staring at a blank screen paralyzed with writer's block): Rachel Nance and Frank and Ellen (and Simon) McCormick.

**The strangers.** Thanks to the cookbook authors who gave me tasks that I could successfully complete in under 2 hours (so unlike research!), the internet people who entertained me when I was lonely and had no friends, and the truck driver who gave me a ride home that one time I was stranded by the side of 295 (no joke, dude saved my life).

**And of course my advisors.** Jim Yorke, Aleksey Zimin, Konstantina Trivisa, and especially for all that life advice and encouragement, Alverda McCoy.

## Table of Contents

|  |      |
|--|------|
| Acknowledgments.....   | iii  |
| List of Tables .....   | vii  |
| List of Figures.....   | viii |
| CHAPTER 1: INTRODUCTION AND BACKGROUND .....   | 1    |
| Part 1.1: Introduction .....   | 1    |
| Part 1.2: Background .....   | 2    |
| Section 1.2.1: A tour through Biology.....   | 2    |
| Section 1.2.2: An Overview of Genome Sequencing .....  | 4    |
| Section 1.2.3 Detection and Correcting Assembly Errors and Omissions: Our Goal. 9                |      |
| Section 1.2.4: The State of Genome Assemblies Today, and Current Methods to<br>Improve Them..... | 12   |
| CHAPTER 2: THE CE STATISTIC .....  | 17   |
| Section 2.1: Theoretical Framework.....  | 17   |
| Section 2.2: Implementation of the Method.....   | 23   |
| Section 2.1 Experimental Evaluation of the CE Statistic.....                                     | 34   |
| CHAPTER 3: THE SHOOTING METHOD and RESULTS.....  | 42   |
| Part 3.1: The Shooting Method.....   | 42   |
| Section 3.1.1: The Shooting Algorithm .....  | 42   |
| Section 3.1.2: Multiple Paths .....  | 50   |
| Part 3.2: Results and Discussion.....  | 52   |
| Section 3.2.1: Bacterial Assemblies.....   | 52   |
| Section 3.2.2: Limitations of the Method .....   | 57   |
| Section 3.2.3: <i>Bos taurus</i> .....   | 63   |



Section 3.2.4: An Assembly of Simulated Short Read Data from Human  
Chromosome 1 ..... 68  
APPENDIX A: THE ALLPATHS ALGORITHM..... 74  
Bibliography ..... 77

## List of Tables

|   |    |
|---|----|
| Table 1: Assemblies with gaps. ....   | 13 |
| Table 2: The size of compressions that can be detected. The <i>spanning size</i> of an insert is the size of the insert minus 200 bp, the total length of the reads at the ends of the insert. This size is used in computing the <i>average spanning insert coverage</i> . Recall that we estimate the minimum size of the compression that can be detected from the formula $(T\sigma w)/N$ . This is an underestimate since the spanning insert coverage decreases at the site of a compression. The upper bound is the spanning size of the insert library. Because of the fluctuations in insert size, occasional compressions outside of the range may be detected..... | 69 |
| Table 3: Statistics of faux assembly of human chromosome 1. Recall that the N50 contig size is the largest contig such that the sum of the number of bases in all larger contigs comprise 50% of the assembly. The original chromosome is the finished sequence. ....   | 70 |
| Table 4: Sensitivity and false discovery rate (FDR) of the CE statistic in detecting expansions for various thresholds $T$ . ....   | 71 |

## List of Figures

|  |    |
|--|----|
| Figure 1: The familiar double helix. ....  | 3  |
| Figure 2: A,B,C represent reads where A overlaps B and C. However, read C does not overlap read B. Their sequence differs on the right side. ....  | 7  |
| Figure 3: Overview of genome assembly. The top line of this schematic symbolically describes the complicated process of finding overlapping reads, leading to contigs. Using mate pairs, one creates a layout, ordering and orienting contigs with gap sizes between contigs labeled d1 and d2. ....   | 8  |
| Figure 4: Resolving small differences in overlapping reads to create consensus sequence. Each horizontal sequence below the blue line represents a read. When aligning reads with each other, allowances must be made for the case where some reads have one or more extra letters. Dashes indicate spaces to show that some other read(s) have additional letters in that location. That can be due to errors in sequencing or actual differences in genomic sequences. ....  | 9  |
| Figure 5: Here Haplotype 1 on top differs from Haplotype 2. In particular, the orange region B is different from the blue B'. In the assembly these two regions lie on either side of the gap, as the assembler could not place these regions on top of each other. If B and B' are less than the length of a read, then the assembler could resolve these differences by choosing a read which covers this location. But if they are larger, the assembler may not be able to choose between the two, and can leave a gap instead. .... | 11 |
| Figure 6: Unweighted and weighted insert size distributions for a library from the rat genome data. This library has $\mu=1978$ , $\sigma=937$ , and $\mu_w=2422$ , $\sigma_w=1061$ . ....   | 24 |

|   |    |
|---|----|
| Figure 7: An example of a compression misassembly. The blue interval in the true genome is omitted in the assembly (bottom line). Inserts 1 and 2 appear compressed in the assembly.....  | 27 |
| Figure 8: Size of the region around location $x$ , where the number of inserts spanning $x$ exceeds the number of inserts which do not span $x$ , is on average $\mu/2$ . .....   | 30 |
| Figure 9: A graph of the CE statistic computed at each insert start position and end position for a simulated assembly with a compression near the 50,000 bp location. ....   | 31 |
| Figure 10: Number of false positives generated by the CE statistic for an assembly made from an ideal normal library. This graph uses $\alpha = 0.5$ , and the two curves cross at $T = 2.1$ . Choosing $\alpha$ is a tradeoff between a good fit at small $T$ and large $T$ . .....  | 35 |
| Figure 11: Histogram of the insert sizes for the 2771 library .....   | 36 |
| Figure 12: Plot of average number of CE problem points found in a perfect simulated assembly for the library with weighted mean 2771. The theoretical model (blue crosses) captures most of the variation in the number of false positives as a function of the threshold $T$ . The model slightly overestimates the actual number of false positives (orange x's) when $T < 2.3$ and underestimates it for $T > 2.3$ . ..... | 37 |
| Figure 13: Plot of sensitivity (red crosses) and FDR (green x's) for different thresholds for the CE statistic using the library with weighted mean 2771. The blue stars represent the difference between the sensitivity and FDR which we would like to maximize. This can aid us in choosing a reasonable range of thresholds for the CE statistic. In this case, a good threshold would be $3.2 < T < 4$ . .....           | 38 |
| Figure 14: Histogram of insert sizes of the library with weighted mean 2422.....  | 39 |

- Figure 15: The sensitivity and FDR of the CE statistic for thresholds  $2 < T < 5$  for a simulated assembly. This assembly consisted of 5 10 Mbp contigs each with alternating 500 bp compressions and expansions every 500,000 bp. The assembly was made with a library with weighted mean 2422. .... 40
- Figure 16: The sensitivity and FDR of the CE statistic for thresholds  $2 < T < 5$  for a simulated assembly. This assembly consisted of 5 10 Mbp contigs each with alternating 1500 bp compressions and expansions every 500,000 bp. The assembly was made with a library with weighted mean 2422. .... 41
- Figure 17: A schematic demonstrating the Shooting Algorithm. The two contigs, contig 1 and contig 2, flank a gap. The orange lines are the target reads chosen from each contig far enough away from the ends to avoid poor quality sequence (grey section of contig). We “shoot” from read 1 to read 2. The blue lines are reads which overlap read 1. They are the beginning of the wave front which will extend from read 1 to read 2. .... 43
- Figure 18: The orange reads represent the reads that make up the wave front as we iteratively build wave 1 from read 1 to read 2. .... 44
- Figure 19: Wave 1 of the Shooting Algorithm. .... 45
- Figure 20: The blue and orange segments denote reads in wave 1. The orange reads are the reads that are in both wave 1 and wave 2. Any path of reads between read 1 and read 2 must be built from the orange reads. .... 45
- Figure 21: A smoothed histogram of a library from an assembly of *Acidobacterium bacteria Ellin345* with  $\mu = 6882$  and  $\sigma = 662$ . Note the double peak. .... 61
- Figure 22: A schematic showing a merged contig containing two “original” contigs (black segments), and new gap sequence produced by the Shooting Algorithm (orange

segment) as well as a now redundant contig placed next to the merged contig on the chromosome. The redundant contig's sequence matches that of the gap sequence, and so it is removed from the assembly. This contig was probably erroneously placed next to the other contigs instead of being placed between them in the original assembly. The sequence is now correctly placed..... 66

**Figure 23: Compressions found using three libraries.** We created faux reads from human chromosome 1 sequence. We created three libraries of mated reads and one library of unmated reads. Using the Celera Assembler 5.4, we created a (draft) assembly, which turned out to have 482 compressions (red circles). We were able to identify 284 of them using one or more of the libraries. The compressions range from 40 bp to 1935 bp (vertical axis). We numbered the compressions, sorted by size, from smallest to largest (horizontal axis). When a red circle indicates a compression that was identified by one of the three libraries, this is indicated by symbols shifted upward (for the sake of clarity) from the red circle. We required that the position and size of the compression be correctly estimated (see text). ..... 71

# CHAPTER 1: INTRODUCTION AND BACKGROUND

## Part 1.1: Introduction

With the completion of the draft of the human genome in 2001 (Venter, et al. 2001), the world's attention has turned to sequencing the genomes of other living things. The human genome was expensive to finish in terms of both cost and time, and as the number of additional organisms we wish to sequence grows, we are trying to find cheaper and faster ways of creating a quality draft of the genome's sequence called a *genome assembly*.

Virtually all assemblies produced today contain errors and omissions in the sequence. For example, the human genome has been worked on far more extensively and with great expense out of any genome currently sequenced. However, even human chromosome 1 (which we analyze later in this dissertation), is missing up to 10% of its sequence. We have no way to count the number of errors. The goal of our research is to detect and correct certain types of errors and to fill gaps in draft genome assemblies using existing data. Our focus is on developing post-processing techniques which will improve an existing draft assembly without needing any additional sequencing or data from other assemblies (of the same or closely related organisms). Our hope is this will decrease the cost, both in time and money, of assembling and finishing genomes.

The rest of this chapter provides a brief introduction to the biological concepts necessary for this paper. We also describe current sequencing techniques, and then discuss the dominant assembly programs in use today. We review the current state of genome assemblies being produced, including the prevalence of gaps and certain types of errors in the sequence. We then describe the motivation for our methods of detecting

these errors, correcting them, and filling gaps in draft assemblies.

In Chapter 2, we describe our method of detecting certain types of errors in assemblies. We develop a statistic, called the CE statistic which we use to analyze potential (erroneously) inserted and deleted regions in genome assemblies. We then evaluate the use of this statistic using simulated data.

In Chapter 3, we discuss an algorithm called the Shooting Algorithm for filling gaps in assemblies, and describe how we also use this method to improve the detection of errors by the CE statistic and to attempt to correct such errors. We put forward results of implementing and evaluating both the CE statistic and the Shooting Algorithm on 22 bacterial assemblies for which we have high quality finished sequence. We then describe the use of the Shooting method on a de novo assembly of *Bos taurus*. Finally, we present some results of using the CE statistic and the Shooting method on an assembly of human chromosome 1 using faux data created to simulate an actual data set produced using the newer short read technology.

## **Part 1.2: Background**

### **Section 1.2.1: A tour through Biology**

We will first go through a very brief tour of the basic well-known biological concepts we will need for this paper. *DNA* or *deoxyribonucleic acid* is a molecule contained in all living cells. The molecules are in the form of the familiar double helix. Each strand is made up of a chain of molecules called *nucleotides*. Each nucleotide is made up of a sugar-phosphate group and a nitrogenous base. There are four nitrogenous bases (or simply *bases*): adenine (A), thymine (T), cytosine (C), and guanine (G). These bases pair up (into *base pairs*) connecting the two individual strands into one double



helix. The bases pair up in the following way: adenine with thymine and cytosine with guanine. (See figure 1). Because of this one-to-one pairing, if we know the sequence of bases of one strand of the double helix, we will know the sequence of the other strand (called the *complementary sequence*).

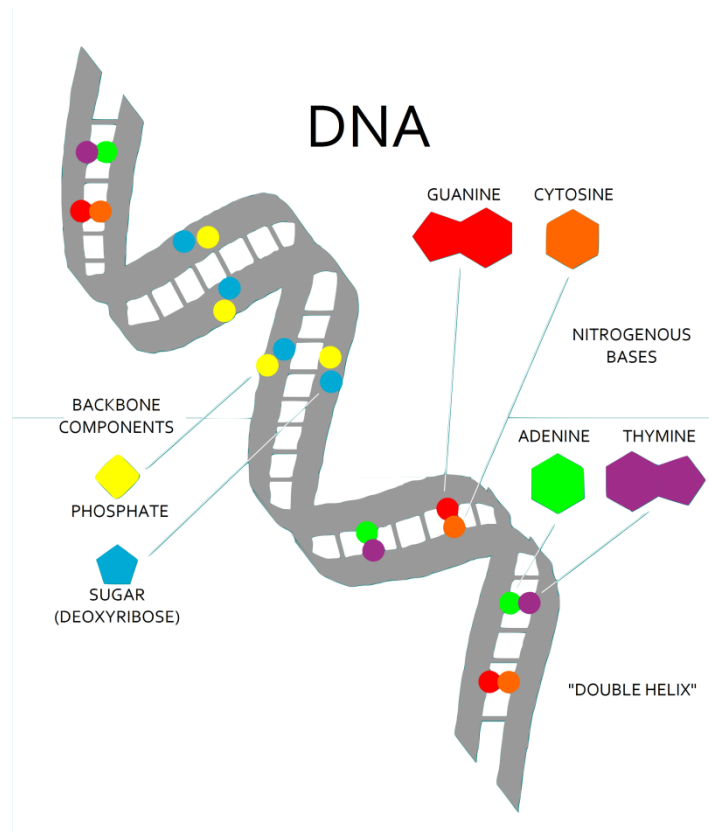


Figure 1: The familiar double helix.

Each chromosome contains one DNA molecule, together with support structures (such as histones). Most bacteria have only one chromosome. Humans have 46 chromosomes, 44 of which come in pairs and 2 of which are sex chromosomes, which are paired only in females. Cows have 29 pairs of chromosomes and 2 sex chromosomes. Organisms whose chromosomes come in pairs are called *diploid*, and the paired chromosomes are called *homologues* of each other. In the case of diploid organisms that

reproduce sexually, each homologue comes from a different parent, one maternal and one paternal. Organisms that have more than 2 sets of chromosomes are called *polyploid*. Polyploidy is fairly common in the plant kingdom including some ferns and flowering plants (for example, *Coffea arabica* is tetraploid). Each homologous chromosome is almost identical to its partner(s) often with only small differences. Variations like these in homologous chromosomes are called *polymorphisms*. The size of these variations can range from single nucleotide polymorphisms (*SNPs*) to longer rearrangements and deletions usually up to 100 bp, but sometimes much longer. The SNP rate in the typical human genome is about 1/1000 (The International HapMap Consortium 2007). These polymorphisms are usually inherited in blocks called linkage groups, and the specific pattern of polymorphisms in a block in a single homologue is called a *haplotype*.

DNA contains all the hereditary information for an organism. This information, as far as we know, can be completely characterized by the sequence of bases in each chromosome. The sequences of bases in each chromosome together comprise what we call a *genome*. It is this information we are interested in finding out, and in the next section we will see how we get this information by sequencing the genome.

### **Section 1.2.2: An Overview of Genome Sequencing**

DNA is one very long molecule, and there is currently no way to read all of the bases of the sequence in a single chromosome all at once in a single process, although research is active in this area. Instead, alternative methods of reading the bases in shorter strings and reconstructing the original sequence are used.

**Sanger Sequencing.** The data we used in our research was produced using Sanger chemistry methods. In this method, DNA is sequenced by first taking many

copies of each molecule. These copies are then cut randomly into pieces usually with mechanical shearing or shock waves. After this random shearing, we do not know where the pieces come from on the chromosome (or to which chromosome they belong). These pieces, called *inserts*, are then sorted by size, and certain size ranges are selected for use in the assembly. As we will see, good estimates of insert sizes as well a good selection of varying sizes is important to an assembly procedure.

The actual base letter sequences of the inserts are read by another laboratory process. This process will read the bases in only one direction, and it may not be able to read every letter along the entire length of each insert. In our data, only the sequence near each end of the insert is read. There usually is a portion of the middle of the insert whose sequence is unknown. The pieces from the ends of each insert for which we have sequence are called *reads*. The lengths of these reads are usually 500-1000 bp (after trimming off low quality or contaminant sequence from the ends of the read). Two reads from each end of the same insert make up a *mate pair*. We know the approximate length of each insert (usually +/- 10%), and thus we can estimate how far apart each read in a mate pair is from the other in the genome.

We should note here that in the case of diploid and polyploid organisms, DNA is taken from all chromosomes, so sequence from all homologues may be represented in the read data. We will have no way of knowing which reads come from which chromosomes.

**Other sequencing technologies: short reads.** While most of the experiments in this thesis used Sanger reads, our ideas can be used with the newer short read data using Illumina and 454 (Shendure et al. 2005), (Margulies et al. 2005) technology. These reads

also come in pairs, but the reads are significantly shorter than those produced in the Sanger process. To test our methods, we need to refer to genome assemblies which are of very high quality, so called “finished genomes.” These are currently available only for assemblies made with Sanger reads. Indeed, most finished genomes are viral or bacterial. Other examples of species with very high quality assemblies include *Drosophila melanogaster* and *Caenorhabditis elegans*. To evaluate our methods, we use read data to create a draft assembly, and then use our techniques to improve the assembly. We then compare our results to the reference assembly to see if in fact our improvements are correct. For this reason, we would like to use only high quality assemblies in our testing. As we shall see, due to the dearth of high quality assemblies using short read data, in order to evaluate our methods on short read assemblies, faux data must be used.

**Major Assembly Algorithms.** There are several different assembly algorithms in use today. Each may use a different method for assembling genomes. We will describe one of the most popular, the overlap-layout-consensus paradigm (OLC). This is used by the Celera Assembler, which we use here in our research, and by other OLC assemblers including ARACHNE (Batzoglou et al. 2002), Atlas (Havlak et al. 2004), PCAP (Huang et al. 2003), JAZZ (Aparicio et al. 2002), TIGR assembler (Sutton et al. 1995), and Phrap. Other newer, non-OLC programs have been developed for short read assembly (Schatz, Arthur L. Delcher, and Steven L. Salzberg 2010) and (Ruiqiang Li et al. 2010).

**Finding overlaps between reads.** We say that two reads overlap if the sequence (consisting of As, Cs, Gs, and Ts) at the end of one read is the same as the sequence at the end of another (see Figure 2). The first step in OLC assembly after trimming is

usually to find all overlaps between reads.

**Contigs.** The Celera Assembler uses these overlaps to produce longer pieces of sequence called *contigs* (contiguous sequence) by putting together overlapping reads for which there are few (or no) conflicting overlaps. A conflict can occur as follows: read B can overlap read A on one end, and read C can overlap read A on the same end, but read C does not overlap read B. When these conflicts occur, the assembler stops extending the sequence of overlapping reads, and the contig ends.

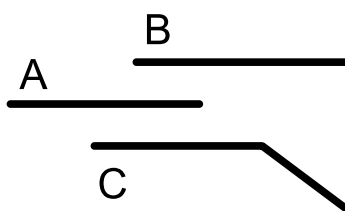


Figure 2: A,B,C represent reads where A overlaps B and C. However, read C does not overlap read B. Their sequence differs on the right side.

In contrast, the ARACHNE assembler looks for what it calls “paired pairs.” Given a mate pair  $a$ , with reads  $a_1$  and  $a_2$ , the assembly algorithm attempts to find another mate pair,  $b$  with reads  $b_1$  and  $b_2$ , such that  $a_1$  overlaps  $b_1$  and  $a_2$  overlaps  $b_2$  in a manner that is consistent with the orientation of the reads in the mate pair and with implied distance between the reads given by the insert length. This constraint that both reads from the mate pair must overlap both reads in its “paired pair,” means that very few of these initial overlaps are incorrect; the paired pair must be entirely contained in a large repeat in order for there to be a problem (base-calling errors excepted). The assembler then repeats the same procedure looking for an insert whose reads  $c_1$  and  $c_2$  overlaps both ends of the previous paired pair. If there is a conflicting overlap on either end (such as  $c_1$  overlaps  $a_1$  and should overlap  $b_1$  but does not), this insert is not added to this collection

of paired pairs. The collections of paired pairs are then merged together to form short contigs (Batzoglou et al. 2002).

**Layout.** With these contigs, the assembler now makes a layout of the assembled sequence. Since the placement of the reads in each contig is known, the assembly program can look for mate pairs for which each read is in a different contig. Since we know the estimated length of each insert, we can estimate how far apart these contigs are and their relative orientation. This information gives us an arrangement of the contigs in relation to each other. These longer sequences of contigs are called *scaffolds*; the contigs may be overlapping in the scaffold or there may be gaps between them. The *layout* is the sequence of contigs in each scaffold along with the placements of all the reads in the contigs.

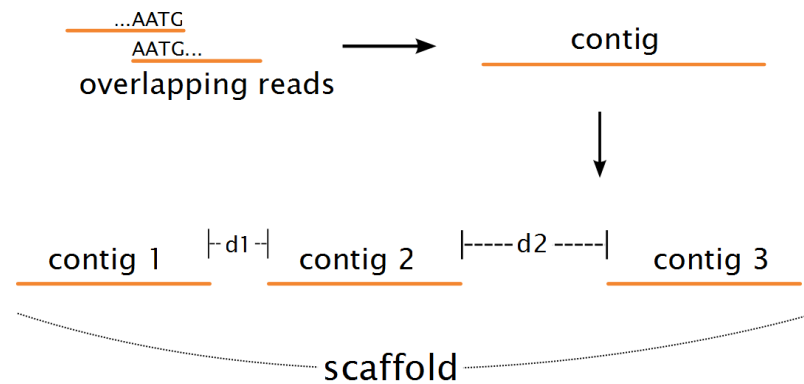


Figure 3: Overview of genome assembly. The top line of this schematic symbolically describes the complicated process of finding overlapping reads, leading to contigs. Using mate pairs, one creates a layout, ordering and orienting contigs with gap sizes between contigs labeled d1 and d2.

**Creating a consensus sequence.** The last step in creating draft assemblies is to produce an actual sequence of letters with gaps that represent most of the genome. Because the quality of the read data is not perfect, we do not require that overlaps between reads be perfect. We may tell the assembly program to allow a small percentage of the overlapping region to have discrepancies between the reads. This results in single

base differences between reads in our contigs. These differences can include substitution of one base for another in a specific single base position, an insertion of an additional base, or the deletion of a base. The assembly program can examine these differences between all the reads that it has placed covering a given location  $x$ , and it can choose which letter occurs with more frequency at  $x$  in the reads. In this way, it can build a final sequence called the *consensus sequence*.

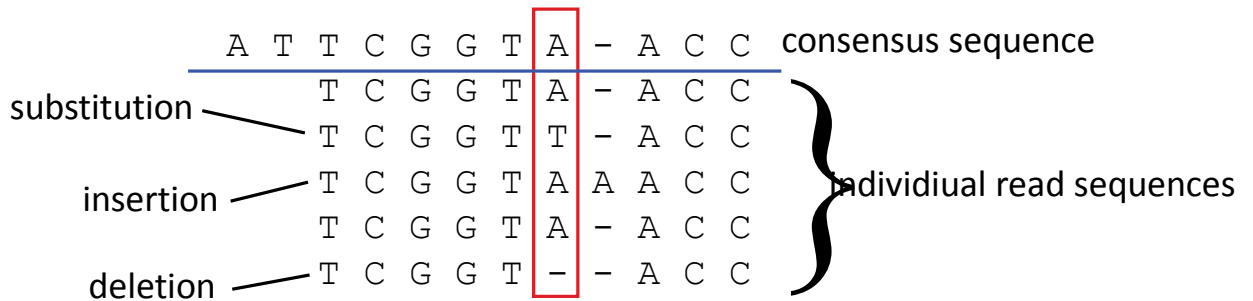


Figure 4: Resolving small differences in overlapping reads to create consensus sequence. Each horizontal sequence below the blue line represents a read. When aligning reads with each other, allowances must be made for the case where some reads have one or more extra letters. Dashes indicate spaces to show that some other read(s) have additional letters in that location. That can be due to errors in sequencing or actual differences in genomic sequences.

### Section 1.2.3 Detection and Correcting Assembly Errors and Omissions: Our Goal.

Creating a draft assembly is not a rigorous process. Ideally, when the assembler makes a choice in the sequence, the choice it makes is overwhelmingly probable. One program may be more conservative, making fewer choices, resulting in a smaller assembly with more gaps. Another program will be more aggressive yielding a draft assembly with bigger contigs and more sequence and possibly more errors. It is easy to see which draft assembly has better statistics, but it is quite difficult to detect assembly errors without doing additional sequencing. Our goal is (1) to detect certain types of errors (see Chapter 2) and (2) close gaps and correct detected errors (see Chapter 3).

We repeat again: A draft assembly is imperfect and can in fact contain thousands

of omissions and uncounted errors. Our aim is to correct two classes of deficiencies: (1) misassemblies resulting from erroneous insertion or deletion of sequence in the assembly and (2) gaps between contigs in the assembly.

We call the first class of deficiencies *compression/expansion or CE misassemblies*. A *compression* occurs when there is a portion of the genomic sequence missing from the interior of a contig in the draft assembly. An *expansion* is when the opposite occurs, and there is sequence erroneously inserted in a contig that is not found in that corresponding position in the genome. These types of errors can result from:

1. errors made by the assembly algorithm;
2. a region of the DNA being uncloneable;
3. poor quality read sequence; repetitive sequence in the genome; or

An example of a repeat region causing an error would be if there is a part of the genome with sections A, R, B, and C which occur as follows: A R B R C, the assembler may compress the region, and produce the sequence, A R C, completely omitting the B section. Alternately, there could be another part of the genome that is laid out as A R C, and our assembler could expand the region to be A R B R C. The assembler could also get confused in this region, and leave a gap: A – C.

The second class of shortcoming we wish to correct in draft assemblies is gaps between contigs in the same scaffold. Closing gaps in genome assemblies is one of the steps towards upgrading the draft assembly to a finished one (Celniker et al. 2002). Also, more of the genes will lie entirely contigs. Traditional gap closing techniques involve creating new sequence data using targeted sequencing and primer walking. These techniques are expensive and time-consuming. We find that in some cases there is



enough information in the original read data to properly close a gap. The assembly program simply could not figure out the correct way to assemble a region.

There are four major reasons for gaps in genome assemblies:

1. the assembly program may incorporate a poor quality read, so the program may not be able to extend the assembly beyond that read;
2. polymorphisms can cause gaps in the regions where the two haplotypes differ because reads from different chromosomes may conflict and contigs on the two sides of the gap can represent different haplotypes of the same region (see Figure 5);
3. repetitive sequence can lead to ambiguities, where there are multiple ways to assemble a region causing the assembler to leave a gap; and
4. there may be no reads that cover a part of the genome.

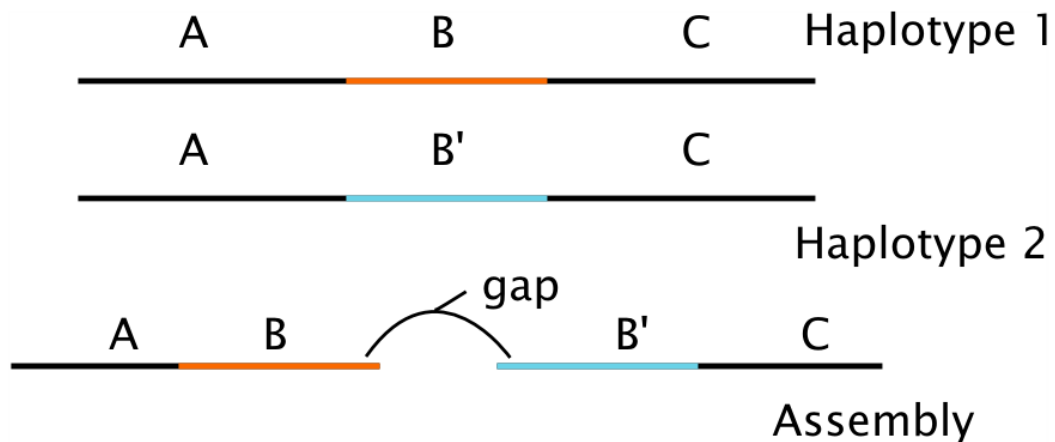


Figure 5: Here Haplotype 1 on top differs from Haplotype 2. In particular, the orange region B is different from the blue B'. In the assembly these two regions lie on either side of the gap, as the assembler could not place these regions on top of each other. If B and B' are less than the length of a read, then the assembler could resolve these differences by choosing a read which covers this location. But if they are larger, the assembler may not be able to choose between the two, and can leave a gap instead.

The goal of our method is to use existing data to close gaps that are due (1), (2), and (3). Gaps that are due to (4) require additional read data. While we may not be able

to provide a definitive assembly for some gaps that are due to repetitive sequence (3), our goal is to report all possible assemblies for that region. For our research, we would like to identify all misassembled and omitted regions in our assembly, find all possible ways to assemble these parts, and identify situations in which there is a single acceptable way. Our method partially finishes the genome and will reduce the cost of traditional finishing.

In the next section we will discuss how prevalent these types of errors and omissions are in draft assemblies as well as other methods of assembly or correcting errors that are currently in use.

#### **Section 1.2.4: The State of Genome Assemblies Today, and Current Methods to Improve Them**

To examine the prevalence of gaps in genome assemblies, we surveyed a collection of genomes which were deposited in GenBank. There were 994 bacterial genomes. As we would expect, due to their small size and the relatively low amount of repetitive sequence of these genomes, there were very few gaps in these assemblies. There were only 33 assemblies which had a total of 157 gaps. As we progress to larger genomes, the picture changes slightly. For example, there were 23 fungal genomes in GenBank. Of these, 14 contained gaps in sequence. There were 258 gaps in all of the assembled fungal genomes.

As genomes grow in size and complexity, we can see that the number of gaps grows as well (see Table 1).

| Assemblies with gaps | (Mean) Length of Genome | Number of Gaps |
|----------------------|-------------------------|----------------|
| 33 Bacteria          | 3.658Mb                 | 157            |

|  |         |         |
|--|---------|---------|
| 14 Fungi                                   | 16.56Mb | 258     |
| Beetle ( <i>T. castaneum</i> )             | 187.5Mb | 5,405   |
| Mosquito ( <i>A. gambiae</i> )             | 230.5Mb | 6,302   |
| Grape Vine ( <i>V. vinifera</i> )          | 303.1Mb | 7,715   |
| Platypus ( <i>O. anatinus</i> )            | 437.1Mb | 45,223  |
| Wild Boar ( <i>S. scrofa</i> )             | 813.0Mb | 54,957  |
| Zebra Finch ( <i>T. guttata</i> )          | 1.021Gb | 59,924  |
| Chicken ( <i>G. gallus</i> )               | 1.032Gb | 53,173  |
| Zebrafish ( <i>D. rerio</i> )              | 1.277Gb | 35,420  |
| Hydra ( <i>H. magnipapillata</i> )         | 1.279Gb | 103,809 |
| Purple Sea Urchin ( <i>S. purpuratus</i> ) | 1.840Gb | 175,267 |
| Dog ( <i>C. familiaris</i> )               | 2.445Gb | 23,037  |
| Chimpanzee ( <i>P. troglodytes</i> )       | 3.188Gb | 224,294 |
| Opossum ( <i>M. domestica</i> )            | 3.502Gb | 53,825  |

Table 1: Assemblies with gaps.

It is important to note that the gaps reported are only supposed gaps. There, in fact, may be no missing sequence in these assemblies; the assembly algorithm used may have erroneously put a gap there. The adjoining contigs may actually overlap. Sequencing centers are simply reporting regions of their assemblies where there is an ambiguity.

**Examples of compressions and expansions in other assemblies.** We now describe other studies that have shown that there are compression and expansion errors as

well in genomes considered finished. A study (Deshayes et al. 2007) on the bacterial genome *Mycobacterium smegmatis* identified regions with possible errors by comparing the sequence to other reference bacterial genomes. Researchers then resequenced these regions and found that there were 28 sequencing errors (compressions and expansions) which resulted in finding that 18 previously predicted genes do not actually exist in this species, and one new gene exists which was previously unknown for this organism. Another recent study (Mandel, Stabb, and Ruby 2008) of a microbial genome, *Vibrio fischeri*, found that there existed 174 individual compression, expansion, or substitution misassemblies, which were corrected by the study's researchers. This included 14 large errors ranging in size from 318 bp to 1264 bp. All of these errors affected suspected gene locations in the genome. An earlier letter (Steven L. Salzberg and Yorke 2005) cited compressions in the *Drosophila* assemblies where the deleted regions could be quite long, on the order of 5000-10,000 bp, and in some cases such misassemblies can total 1% of the whole genome.

The previous examples highlight the real reason we would like to correct gaps and errors in genomes. While the actual amount of sequence we are correcting may be small, it may come from scientifically important parts of the genome, such as regions containing genes or regulatory information. We will now see how assemblers currently deal with gaps, and also how other post-processing methods attempt to fill these gaps and correct other misassemblies.

**Other methods for correcting misassemblies and gaps.** The assembly algorithms we mentioned in the previous sections have methods of filling gaps between contigs within a scaffold. The ALLPATHS approach is similar to the methods we

present in our research, but there are a few key differences. A fuller treatment of these differences can be found in the Appendix.

The ARACHNE assembler looks for a path of overlapping contigs (from a list of not yet used or repeat contigs) that will fill the gap. It first constructs a graph where the nodes are contigs and an edge connects two nodes if the contigs overlap. Then the shortest path between any two contigs in the graph is recorded (within reason – path lengths are restricted by a threshold). If any two contigs flanking a gap in the scaffold have a shortest path, then this gap is filled in. Otherwise, the assembler looks for contigs that can be placed in the gap using mate pairs, and then again looks to see if there is a recorded shortest path between any of these contigs.

The Celera Assembler has two main methods to deal with gaps. The first utilizes mate pair information to fill in a gap. It looks for an unplaced or repetitive contig, which contains a read whose mate is placed in a contig flanking the gap. Using information about insert lengths and the orientation of the placed read, the algorithm can order and orient this unplaced contig in the gap. The contig will be placed there depending on the number of mate pair links which infer this location. The algorithm can also repeat this step looking for contigs that could potentially lie between this newly placed contig and the flanking one.

The other way in which the Celera Assembler attempts to close (or at least reduce in size) gaps in the draft assembly is to attempt to extend the flanking contigs into the gap. It does this by relaxing some of their error thresholds for overlaps and for quality of sequence, and then looking for overlapping reads that could extend the contigs into the gap.

Another approach (which we use in our research) is to create a post-processor to close gaps left by an assembly program. A current method (Zimin et al. 2008) for closing gaps and identifying and correcting misassemblies involves comparing the sequence to other assemblies of the same organism, which were assembled in different ways. Similarly, the increasing number of related genomes being assembled means that new draft assemblies can also be aligned to closely related species in order to fill in gaps or correct potential errors. Still, the gold standard method is to resequence problematic portions of the genome in the wet lab using biochemical techniques, which is a costly and time consuming process.

Ideally, the assembly of a genome evolves over time. Re-assembly, resequencing and re-annotation of assemblies can continue; genomes are rarely completely finished. Our goal is to help these efforts by closing gaps and identifying and correcting compression/expansion misassemblies without resorting to resequencing to garner additional data. As we shall see in Chapter 2, our method for detecting compressions and expansions in draft assemblies is very different from those employed by other algorithms and software.

Additionally, in Chapter 3, we will show how our procedure for closing gaps is different from the algorithms used by the major assemblers. So when there are gaps or misassemblies in the draft assemblies made by these algorithms, we could provide some improvement.

## CHAPTER 2: THE CE STATISTIC

In this chapter, we develop the *CE statistic* to use in detecting compressions and expansions. In Chapter 3, we will combine this method with another method called the Shooting Algorithm in order to further refine our detection and attempt to correct such errors.

### Section 2.1: Theoretical Framework

**CE misassemblies.** As was previously discussed, compression and expansion misassemblies occur when portions of sequence are erroneously missing or inserted in a contiguous part of the draft assembly. Our goal is to identify where such errors occur. To do this we examine the placement of mate pairs in the assembly.

**The CE statistic.** Recall from the introduction that when DNA is sequenced, we have many copies of the same DNA, which are then randomly cut into inserts. Inserts are collected into size-selected libraries. Let  $\mathcal{L}$  be a library with mean insert length  $\mu$  and standard deviation  $\sigma$ . A position  $x$  in a contig will be *spanned* by an insert if the two reads at the ends of the insert lie in the contig and  $x$  lies strictly between the two reads. Let  $N$  be the number of inserts (from library  $\mathcal{L}$ ) spanning the position  $x$ . We can compute the lengths of all the inserts spanning  $x$  from the positions of reads in the assembly. Let  $M (= M_x)$  be the mean length of the inserts which span  $x$ . We can compare the sample mean  $M$  to the population mean  $\mu$  for the library. We make this comparison using a statistic that we call the CE statistic (for *Compression-Expansion statistic*). Formally, the CE statistic  $Z$  is the Z-score of the sample of inserts:

$$Z = \frac{M - \mu}{\sigma/\sqrt{N}}.$$

**A weighted mean for a library for use in the CE statistic.** Using the mean of the library as the  $\mu$  in the CE statistic raises two issues which can yield biased results. The main issue is that within a single library, inserts longer than  $\mu$  cover more bases than shorter inserts. Therefore, these longer inserts will be overrepresented in the sample of inserts covering the location  $x$ , and the expected value of the lengths of this sample will be longer than the library mean. The second issue is that close to the end of a contig, the inserts that span  $x$  and lie completely in the contig will be shorter than  $\mu$ . To compute the correct mean length of the inserts spanning a particular location  $x$ , we will first develop a formal set of assumptions.

To avoid annoying problems at the ends of chromosomes, we will assume a circular genome in which the size of the genome is larger than any insert size. For mammalian genomes, the genome size is about 10,000 times the length of the largest inserts used in assemblies called *Bacterial Artificial Chromosomes* (BACs). Let  $GS$  be the number of base pairs in the genome. We denote individual bases by the integers  $1, 2, 3, \dots, GS$ , though we could equivalently denote them by the integers *mod*  $GS$  to reflect the circularity of the genome. We write  $A$  for a subset of the genome (which represents the assembled portion of the genome) and connected components of  $A$  are called *contigs*.

**Assumption XU:** Let  $\Lambda$  be the set of all possible insert lengths, i.e.  $\Lambda = \{ l \mid l \text{ is a positive integer, } l_{min} \leq l \leq l_{max} \}$ ; here  $l_{min}$  and  $l_{max}$  are positive integers. We define a probability measure  $\pi(l)$  on  $\Lambda$  which is the probability that an insert has length  $l$ . Let  $\mu$  be the expected value of  $l \in \Lambda$ , and let  $\sigma$  be the standard deviation of  $l$ . Let  $S$  be the set of all possible starting positions of inserts in the genome,  $S = \{ s \mid s \in \{1, 2, \dots, GS\} \}$ . Inserts are selected at random from the product space  $\Omega = \Lambda \times S = \{ (l, s) \mid l \in \Lambda; s \in S \}$ , where



the starting position  $s$  is selected at random from a uniform distribution on  $S$  and  $l$  is selected according to  $\pi$ . We define a measure  $Q$  on  $\Omega$ , where  $Q(l)$  is the probability that an insert of length  $l$  starts at position  $s$ ;  $Q(l)$  is independent of  $s$ . Let  $\Phi$  be the number of inserts selected for the genome.

Let the integer  $x > 0$  denote the position in bases from the beginning of the contig. Let  $\mu_x$  be the expected value of the length of an insert subject to the conditions that it lies completely in the contig and spans  $x$ . In particular,  $\mu_x$  is undefined when  $x$  is contained in no inserts that lie completely in the contig.

To simplify notation, we assume reads are very short compared to the insert length, so short that we assume they have length 0.

The following proposition relates  $\mu_x$  and  $\mu$  when a point  $x$  in a contig is far from the ends of the contig.

**Proposition X0:** Assume XU. Assume the interval  $J = [x-l_{max}, x+l_{max}]$  lies in a contig. Then

$$\mu_x = \mu_w \stackrel{\text{def}}{=} \mu + \frac{\sigma^2}{\mu}.$$

Proof:

From the definition of  $J$ , it follows that every insert spanning  $x$  lies entirely in the contig of  $x$ . The probability that a random insert of length  $l$  spans  $x$  is  $l/GS$ . Then the probability that an insert spans  $x$  is

$$\sum_{l \in \Lambda} \left( \frac{l}{GS} \right) (\pi(l)).$$

Then,

$$\begin{aligned}\mu_x &= \sum_{l \in \Lambda} l \frac{\binom{l}{GS}(\pi(l))}{\sum_{l \in \Lambda} \binom{l}{GS}(\pi(l))} \\ &= \sum_{l \in \Lambda} \frac{l^2 \pi(l)}{\sum_{l \in \Lambda} l \pi(l)}\end{aligned}$$

From here we can see that,

$$\begin{aligned}\mu_x &= \frac{1}{\mu} \sum_{l \in \Lambda} l^2 \pi(l) \\ &= \frac{E(l^2)}{\mu}\end{aligned}$$

(where  $E(l^2)$  is the expected value of  $l^2$  with respect to the probability distribution  $\pi$ ).

$$= \frac{\sigma^2 + \mu^2}{\mu} = \frac{\sigma^2}{\mu} + \mu.$$

□

In Assumption XU, we introduce a probability distribution of lengths  $\pi$ . There is another probability distribution  $g(l) = l\pi(l)/\mu$  (see above) which we will refer to as the *weighted probability distribution*. Then  $\mu_w$  is the mean of this distribution.

The above proposition analyzes the case where  $x$  is far from the ends of a contig.

We now remove that assumption.

**Proposition X1:** Assume XU. Assume the interval  $J = [x, x + l_{max}]$  lies in a contig. If  $1 < x < l_{min}$ , then  $\mu_x = \mu$ . For  $1 < x < l_{max}$ ,  $\mu_x$  is monotonically increasing in  $x$  (as long as  $x + l_{max}$  is in the contig).

Since we have assumed reads have length 0,  $\mu_x$  is defined when  $x > 1$ . More generally for the result to be true, we should say  $\mu_x$  is monotonically increasing in  $x$  where  $\mu_x$  is defined.

Proof:

If  $1 < x < l_{min}$ , then every insert which starts in the interval  $[1, x]$  will span  $x$ . The expected length of these inserts will be the expected length of the inserts in  $\Lambda$ , and thus  $\mu_x = \mu$ .

Let  $P(y)$  be the probability that an insert length is greater than  $y$ , for  $y \geq 0$ , i.e.  $P(y) = \sum_{l \in \Lambda, l > y} \pi(l)$ . Let  $H(y)$  be the expected value of all insert lengths greater than  $y$ ;  $H(y) = \sum_{l \in \Lambda, l > y} (l\pi(l) / \sum_{l > y} \pi(l))$ . Then, given  $x > 1$ , the expected length of all inserts which start at the first base of the contig and span  $x$  is  $H(x)$ . In particular, we are interested in the expected length of all inserts which start between the beginning of the contig and  $x$  which are also long enough to span  $x$ . Let  $m > 0$  be the expected number of inserts starting at each point  $x$ . The expected number of inserts starting at position  $x - y$  that span  $x$  is  $m \sum_{l \in \Lambda, l > y} \pi(l) = mP(y)$  and the expected length of these is  $H(y)$ . Thus,

$$\begin{aligned} \mu_x &= \frac{\sum_{y=1}^{x-1} H(y)mP(y)}{\sum_{y=1}^{x-1} mP(y)}, \\ &= \frac{\sum_{y=1}^{x-1} H(y)P(y)}{\sum_{y=1}^{x-1} P(y)}, \quad (1) \end{aligned}$$

And,

$$\mu_{x+1} = \frac{H(x)P(x) + \sum_{y=1}^{x-1} H(y)P(y)}{P(x) + \sum_{y=1}^{x-1} P(y)}. \quad (2)$$

Note that

$$H(x) \geq \frac{\sum_{y=1}^{x-1} H(y)P(y)}{\sum_{y=1}^{x-1} P(y)}$$

This is because the expected length of an insert given that the length is greater than  $x$  is at least as large as the expected lengths of the inserts with lengths greater than  $y$  ranging from 1 to  $x-1$  ( $H$  is monotonically increasing). Then,

$$\frac{H(x)P(x)}{P(x)} \geq \frac{\sum_{y=1}^{x-1} H(y)P(y)}{\sum_{y=1}^{x-1} P(y)} = \mu_x. \quad (3)$$

We use the following fact:

$$\text{For } A, B, C, D > 0, \frac{A}{B} \geq \frac{C}{D} \text{ implies } \frac{(A+C)}{(B+D)} \geq \frac{C}{D} \quad (4)$$

Inequality 3 has the form  $\frac{A}{B} \geq \frac{C}{D}$ . While 2 has the form  $\frac{(A+C)}{(B+D)}$  for the same  $A, B, C, D$ . Therefore, inequality 4 implies  $\mu_{x+1} \geq \mu_x$ . Therefore,  $\mu_x$  is monotonically increasing in  $x$ .

□

**Variance of the sample of inserts spanning  $x$ .** In the previous section, we stated our assumptions and derived the expression for  $\mu_w$ , the expected length of an insert spanning a location  $x$  in a contig. We now derive the expression for  $\sigma_w^2$ , the variance of the weighted probability distribution  $g$  defined above. Write  $E_g(Y)$  and  $E_\pi(Y)$  for the expected values of  $Y$  with respect to probability distributions  $g$  and  $\pi$  respectively.

$$\textbf{Proposition X2} \quad \sigma_w^2 = \sigma^2 \left(1 - \frac{\sigma^2}{\mu^2}\right) + \frac{E_\pi((l-\mu)^3)}{\mu}.$$

Proof:

$$\sigma_w^2 = E_g((l - \mu_w)^2) = \sum_{l \in \Lambda} (l - \mu)^2 g(l)$$

$$\begin{aligned}
&= \sum_{l \in \Lambda} \left( (l - \mu_w)^2 \frac{l\pi(l)}{\sum_{l \in \Lambda} l\pi(l)} \right) \\
&= \frac{E_\pi((l - \mu_w)^2 l)}{E_\pi(l)} \\
&= \frac{E_\pi(l^3 - 2l^2\mu_w + l\mu_w^2)}{E_\pi(l)} \\
&= \frac{E_\pi(l^3) - 2\mu_w E_\pi(l^2) + \mu_w^2 E_\pi(l)}{E_\pi(l)} \\
&= \frac{E_\pi(l^3)}{\mu} - \mu_w^2 \quad (5)
\end{aligned}$$

The skewness of  $\pi$  is:

$$E_\pi((l - \mu_w)^3) = E_\pi(l^3) - 3\mu E_\pi(l^2) + 3\mu^2 E_\pi(l) - \mu^3$$

Therefore,

$$E_\pi(l^3) = \mu(3\sigma^2 + \mu^2) + E_\pi((l - \mu)^3)$$

where  $\sigma$  is the standard deviation of the distribution  $\pi$ . Then, substituting for

$E_\pi(l^3)$  in 5, yields

$$\sigma_w^2 = \sigma^2 - \frac{\sigma^4}{\mu^2} + \frac{E_\pi((l - \mu)^3)}{\mu}$$

as claimed.

□

We note the similarity in the form of  $\mu_w = \mu(1 + \frac{\sigma^2}{\mu^2})$  and  $\sigma_w^2 = \sigma^2(1 - \frac{\sigma^2}{\mu^2})$  when probability distribution  $\pi$  has zero skewness, i.e.  $E_\pi((l - \mu)^3) = 0$ .

## Section 2.2: Implementation of the Method

**Determining each library's distribution of insert lengths.** The sequencing center that produces a library provides an estimated mean,  $\mu_{\text{est}}$ , and standard deviation,  $\sigma_{\text{est}}$ , of all the insert lengths in the library. These estimates are educated guesses. To improve upon these estimates, we take all contigs whose lengths are much longer than  $\mu$ . We examine all of the inserts for which both reads are placed in the same contig, avoiding those that are placed completely within  $\mu_{\text{est}} + 3\sigma_{\text{est}}$  from the ends of the contig because inserts that are placed close to the contig's edge are often shorter than average, and including them in a sample can bias our estimates. We then compute the length of each selected insert based on read placements in the contig, (and discard lengths that are clear outliers). From this collection of lengths, we compute the probability distribution of lengths and refer to it as the *library distribution*. The mean,  $\mu$ , and standard deviation,  $\sigma$ , of this distribution are often significantly different from the reported library statistics.

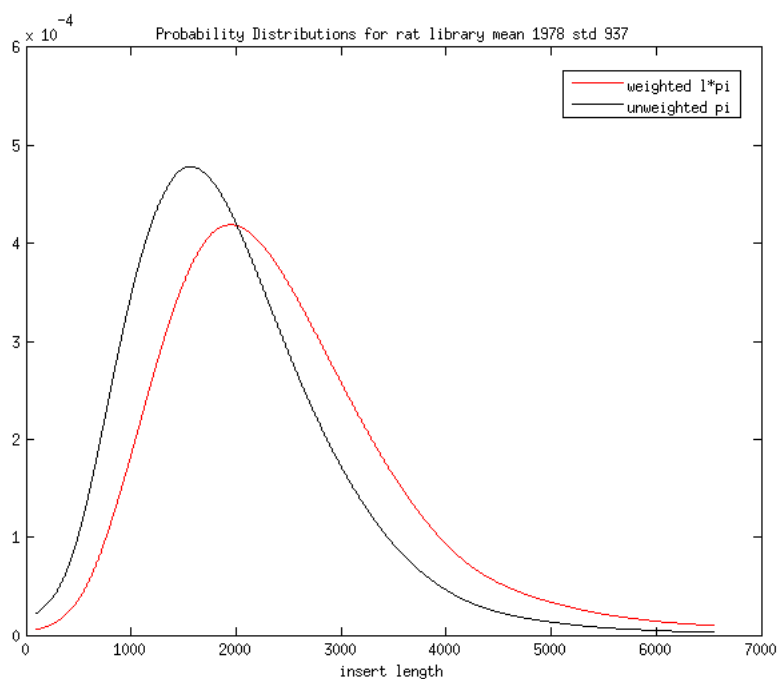


Figure 6: Unweighted and weighted insert size distributions for a library from the rat genome data. This library has  $\mu=1978$ ,  $\sigma=937$ , and  $\mu_w=2422$ ,  $\sigma_w=1061$ .

**The weighted mean in practice.** In practice, we are given a library consisting of  $n$  inserts with lengths  $l_j$  for  $j = 1$  to  $n$ . This collection defines the probabilities  $\pi(l)$  of each length  $l$ . So,

$$\mu_w = \frac{\sum_{j=1}^n l_j^2}{\sum_{j=1}^n l_j}$$

where  $l_j$  is the length of insert  $j$  in the library and  $n$  is the number of inserts in the library. This is the formula derived in Proposition X0 for a point  $x$  away from the end of a contig using  $E(l) = \frac{1}{n} \sum_{j=1}^n l_j$ . From proposition X1, we know that  $\mu_x$  is monotonically increasing in  $x$ , so the minimum value for  $\mu_x$  will be  $\mu$ , and therefore the largest bias we will have using  $\mu_w$  is on the order of  $\sigma^2/\mu$  near the end of a contig. To avoid this bias, we do not compute the CE statistic in regions within  $\mu$  of the end of a contig.

We assume throughout that  $n$  is large, so we can compute the weighted standard deviation as:

$$\sigma_w = \sqrt{\frac{\sum_{j=1}^n l_j (l_j - \mu_w)^2}{\sum_{j=1}^n l_j}}$$

These values are then used to compute the CE statistic,

$$Z = \frac{M - \mu_w}{\frac{\sigma_w}{\sqrt{N}}}$$

With this weighting of the mean and standard deviation, the distribution of the values  $Z$  of the CE statistic approaches *Normal*(0,1) as average coverage increases.

The value  $M - \mu_w$  gives us an estimate for the size of the compression or

expansion. For reasons which will be outlined below, this estimate is not always accurate, and therefore in practice, as we shall see, we use the standard deviation of this estimate ( $\sigma_w/\sqrt{N}$ ) in evaluating our CE statistic.

**The Composite Mean for Multiple Libraries.** Most assembly projects have more than one library. Each point can be spanned by inserts from multiple libraries. In theory, in computing the CE statistic,  $Z$ , with multiple libraries, we would compare a composite sample mean of the insert lengths spanning a location to the composite library mean. The weighting which would minimize the standard deviation of the mean over the different libraries is (Wild 2000):

$$w_i = \frac{(1/\sigma_{wi}^2)}{\sum_{j=1}^N (1/\sigma_{wj}^2)}$$

where  $w_i$  is the weight given to the  $i$ th insert spanning  $x$ , and  $\sigma_{wi}$  is the weighted standard deviation of the library containing insert  $i$ . Then the new composite sample mean is

$M_{w*} = \sum_{i=1}^N w_i l_i$ . We could similarly weight the combined library mean which would be  $\mu_{w*} = \sum_{j=1}^{n*} w_j l_j^2 / \sum_{j=1}^{n*} w_j l_j$ , where  $n^*$  is the total number of inserts in all libraries. In practice, there is little gain in resolution of CE misassemblies using this composite approach. Instead, we identify CE misassemblies using individual libraries.

**Detecting compressions and expansions.** If there is a compression in location  $x$  in the assembly, we would expect, on average, the inserts spanning  $x$  to be shorter than  $\mu$ , and therefore location  $x$  would have a negative CE statistic. Conversely, if there is an expansion, the sample mean  $M$  would be greater than  $\mu$  resulting in a positive CE statistic. In this way, the CE statistic can suggest there is a misassembly in a certain region and provide an estimate of how much the region may have been compressed or



expanded. Figure 7 shows an example of a compression misassembly whereby a section of sequence, shown in blue, has been erroneously omitted from the assembly. This compression causes inserts 1 and 2 to appear shorter in the assembly.

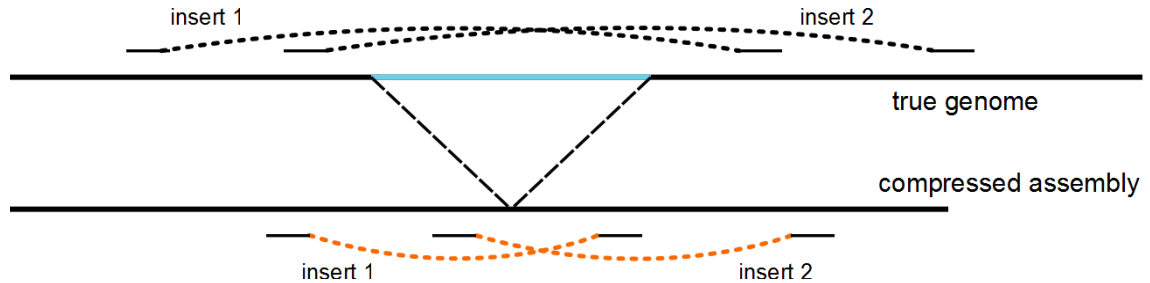


Figure 7: An example of a compression misassembly. The blue interval in the true genome is omitted in the assembly (bottom line). Inserts 1 and 2 appear compressed in the assembly.

In order to use the CE statistic to detect possible misassemblies, we need to choose a threshold  $T$  and define each point  $x$  to be a *CE problem point* when  $|Z| > T$ . These points are near suspected locations for compression or expansion misassemblies. The size of the CE misassemblies we can detect is dependent on the insert library statistics and, of course,  $T$ . The length of a compression that can be detected in practice is approximately bounded above by the library mean  $\mu$ , since a longer region that is omitted will have no inserts which span it. The lengths of both compression and expansion misassemblies are bounded from below by  $(T\sigma)/\sqrt{N}$ .  $Z$  can be large due to a small fraction of the inserts spanning the point being very large while the majorities are of expected length. Such outliers should be ignored.

**CE misassemblies can cause reduced spanning insert coverage and underestimation of the CE problem size.** In computing the sample mean,  $M$ , the CE statistic uses the average insert length of all inserts spanning a point  $x$ . In order to compute the lengths of these inserts, both mates must be placed in the assembly.

However, when there is a compression, one of the mates may be missing, as it would normally have been placed in the omitted section of the assembly. Therefore  $N$ , the number of inserts spanning  $x$ , decreases near the site of a compression.

Another consequence of the compression is that the inserts that span it will be those inserts which are long enough to span the compression. Shorter inserts will not be represented in the sample. If the compression is large relative to library mean, then  $\mu_w - M$  will be less than the true size of the compression, since  $M$  will be the average of the longer inserts at the site of the compression.

As an illustration of these phenomena, we used a library from an actual set of data used to assemble the rat genome to show how misassemblies affect spanning insert coverage and the sample mean  $M$  at the site of a compression. This library had a weighted mean of 2422 bp, and a weighted standard deviation of 1061. We chose this library because it was far from being normally distributed, and it had a large standard deviation relative to the mean (see figure 8), so the effects we described above would be more pronounced. It should be noted that libraries such as these can be common in sequencing projects. Before continuing with this example, we introduce some notation.

Suppose there is a compression at a location  $x$  in the assembly. Let  $C$  be the size of the compression, and let  $M$  be the (compressed) sample mean of inserts which span the compression. To estimate  $M$ , we will use the library distribution. We first shift the insert lengths in this library by  $C$ , such that  $l_{compressed} = l - C$ ,  $l \in \Lambda$ . We need to account for the bias in the sample of inserts spanning the compression towards longer inserts, and the drop in spanning insert coverage. To do this, for  $l - C > 0$ , we weight the probability of each insert by  $(l - C)/l$ , an estimate of the proportion of the inserts which are left after the

compression. We then compute the weighted mean as before using this additional

weight,  $M = \frac{\sum_{j=1}^n (l_j - C)^2}{\sum_{j=1}^n (l_j - C)}$ . This new weighted mean will be the estimate of  $M$ . We would

also like to estimate the drop in spanning insert coverage. We again use  $l_{compressed}$  and the new weighted frequency to compute this value.

For a compression of size  $C = 500$ ,  $M$  is estimated to be 2187 bp, and the spanning insert coverage is 74.7% of the original coverage. We would expect to compute  $M = 1922$ , which is 255 bp shorter than the estimate  $M$ ; hence, the compression size is underestimated by 255 bp. The bias towards longer inserts has skewed the sample mean to be larger. For  $C = 1000$ , the effect is more evident:  $M$  is estimated to be 1936 bp (when, without bias, we would expect it to be 1422 bp, and hence the compressions size is underestimated by 514 bp), and coverage drops to 49.4% of the coverage in a non-compressed region. As this example shows, the CE statistic can be an underestimate, and we should take care in choosing a threshold  $T$  which takes this into account.

As with any probabilistic approach, using the CE statistic to identify CE problem points will result in a number of false positives and false negatives. We will discuss these issues further below.

**Evaluation of the CE Statistic.** A misassembly that results in a CE problem point will most often result in a cluster of such points. In creating an algorithm that searches for problems, we chose to progress monotonically through the assembly. We define a point  $x$  to be a robust problem point if it is followed immediately by two more problem points. We use two measures: (1) the sensitivity and (2) the false discovery rate. Let  $TP$  be the number of *true positives*, the number of CE problems points which correctly identify CE misassemblies, both in type and location. We say a CE problem

region is a true positive if (1) it overlaps the location of the true misassembly in the sequence and (2) if the estimated size of the misassembly is within three standard deviations ( $3\sigma/\sqrt{N}$ ) of the true size. Let  $FN$  be the number of *false negatives*, the number of CE misassemblies that were missed by the CE statistic. Then, sensitivity is defined as:

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

Let  $FP$  be the number of *false positives* which is the number of regions that were incorrectly identified as being CE misassemblies by the CE statistic. Then, the false discovery rate (FDR) is defined as:

$$FDR = \frac{FP}{FP + TP}$$

Ideally, one would like to choose a threshold  $T$  which maximizes the difference between these two, i.e. we would like sensitivity to be large and the false discovery rate to be low.

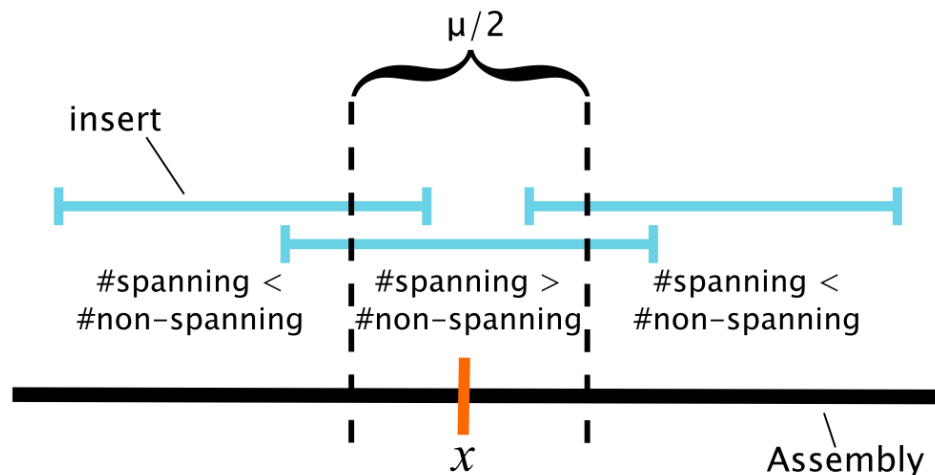


Figure 8: Size of the region around location  $x$ , where the number of inserts spanning  $x$  exceeds the number of inserts which do not span  $x$ , is on average  $\mu/2$ .

**Resolving neighboring CE problem points.** We calculate the CE statistic along

the assembly at all locations where the insert coverage changes, i.e. where an insert begins or ends. In general, in our method of detecting a CE problem point, we are looking for locations  $x$  in the assembly whose CE statistic exceeds the threshold  $T$ . In our implementation, we require the point to be at a peak in the CE statistic, so that the absolute value of the CE statistic decreases on either side of that point. We can detect at most two CE problem points per mean insert length (see figure above). In order to be resolved, the two CE problem points have to have two disjoint sets of inserts covering them. Any inserts that span both points will make it difficult to resolve between the two. If there are several inserts spanning both problems, they will be detected as a single CE problem point. Below, you can see a plot of how the CE statistic varies along one section of an assembly where there is a compression at around 50000 bp.

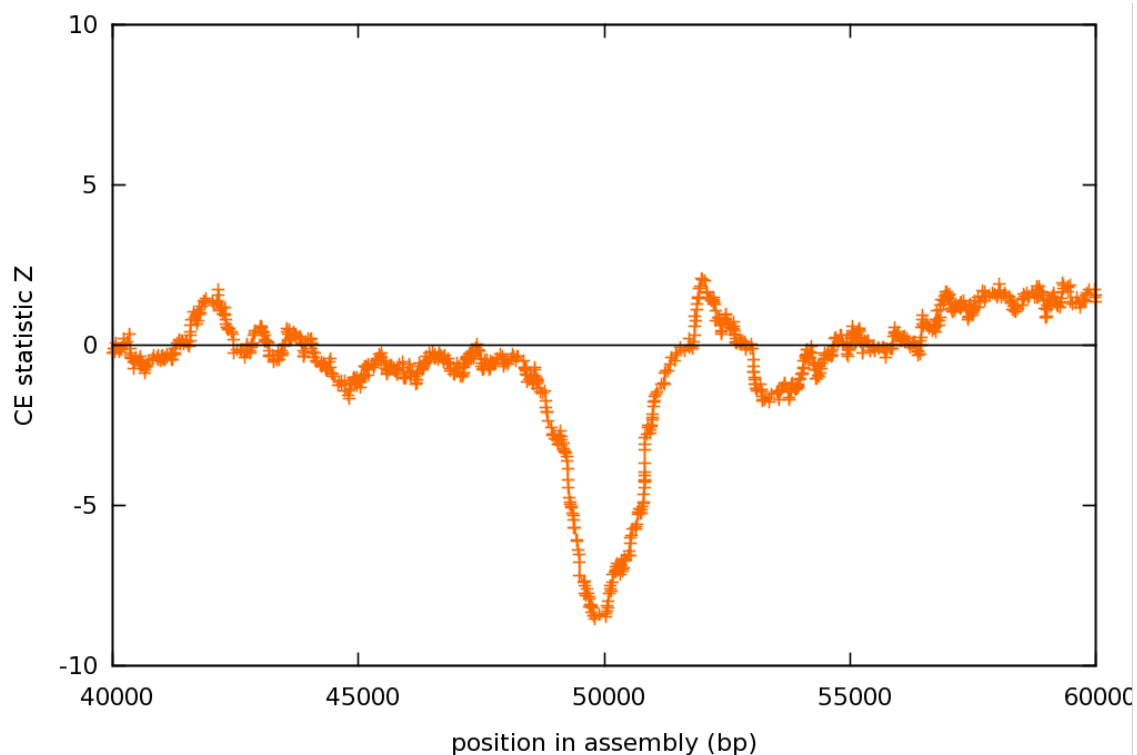


Figure 9: A graph of the CE statistic computed at each insert start position and end position for a simulated assembly with a compression near the 50,000 bp location.

**Estimating the natural rate of CE problem regions.** The goal of this section is to estimate the natural rate of false CE problem regions by studying cases where there are no actual assembly errors. Such problem regions can cause us to miss some true misassemblies. For example, if there is a false positive CE problem point which is an expansion next to (within  $\mu/2$ ) a true compression, this compression may not get detected by the CE statistic.

At each location  $x$ , there is a collection of inserts (from a specified library) that span  $x$ . These inserts have a mean length  $M_x$ , and our goal is to approximate the distribution of these mean lengths using a sampling distribution. While the insert coverage varies with  $x$ , we compute the sampling distribution for a fixed  $N$ , which corresponds to the mean insert coverage, given the library distribution (which we compute as mentioned above at the beginning of this section). We note that for higher insert coverage,  $N > 30$ , according to the Central Limit Theorem this distribution is nearly normal. The sampling distribution allows us to estimate the number of points where the CE statistic is above or below the threshold  $T$ . From the sampling distribution, we can estimate the probability,  $p(T, N)$ , of  $M_x$  deviating from  $\mu_w$  by more than  $T$  standard deviations.

A CE misassembly will cause a number of inserts to be compressed or expanded. Thus, a sufficiently large misassembly will result in the CE statistic being above the threshold in the neighborhood of the CE problem point. The size of this neighborhood varies depending on the size of the misassembly, the threshold  $T$ , and the library mean  $\mu$ . The size of this correlated neighborhood is bounded above by  $\mu$  for compressions.

Let us assume that we have a perfect genome assembly of length  $G$  bp. We can

estimate the natural rate of the CE problem regions by estimating the average size of the CE neighborhood as  $\alpha\mu$ , where  $\alpha$  is a constant. Then the number of such neighborhoods, where the CE statistic is above the threshold can be estimated as:

$$F_1(T) = \frac{p(T, N)G}{\alpha\mu}.$$

Note that some CE neighborhoods may be overlapping especially at low thresholds where CE problem neighborhoods are more numerous. We can correct for this by assuming that the CE problems are distributed along the assembly as if their locations were chosen from a uniform distribution. Then, according to the Poisson distribution, the fraction of the genome covered by CE neighborhoods (Lander and Waterman 1988) is

$$1 - e^{-F_1(T)\alpha\mu/G}.$$

Since the majority of the neighborhoods do not overlap for high thresholds, we can estimate the total number of CE problems as

$$F(T) = \frac{G(1 - e^{-F_1(T)\alpha\mu/G})}{\alpha\mu},$$

that is,

$$F(T) = \frac{G(1 - e^{-p(T, N)})}{\alpha\mu}. \quad (8)$$

Since it is difficult to estimate  $\alpha$  theoretically and in general, we estimate  $\alpha$  by fitting the resulting curve for  $F(T)$  to the simulation. This argument is only intended to describe the behavior of the natural CE problem rate as function of T.

Estimating  $F(T)$  is not an easy problem because we can only estimate the probability of the CE statistic going above the threshold at every point assuming that points are uncorrelated. In fact, as stated above, the values of the CE statistic at

neighboring points are correlated and the autocorrelation varies between 1 and zero over a length between 0 and  $\mu$ , so our estimate of the correlation length of  $\mu/2$  is only that. Therefore, to truly describe the behavior of the CE statistic, we had to simulate the problem, and our theoretical estimate was only intended to roughly approximate this behavior.

## Section 2.1 Experimental Evaluation of the CE Statistic

**Experiments with perfect simulated assemblies.** We created a simulated assembly of five contigs with length 10 Mbp. These contigs were designed to be perfect with no errors. We sampled the insert lengths for the assembly from a normal distribution with  $\mu = 1997$  and  $\sigma = 200$ . Insert coverage for this assembly was on average 50. Each read in the assembly was 100 bp. We would expect the results to be independent of read size as long as the size of the misassembly is longer than the read size. Shorter misassemblies would be rare. We then computed the CE statistic for this assembly. Below, we can see the average number of CE problem points detected per contig, as well as the estimated number we should expect using the formula  $F(T)$ . Here, we fit the estimated curve to the data using  $\alpha = 0.5$ . Since our assembly is perfect, all of these problem points can be considered false positives.



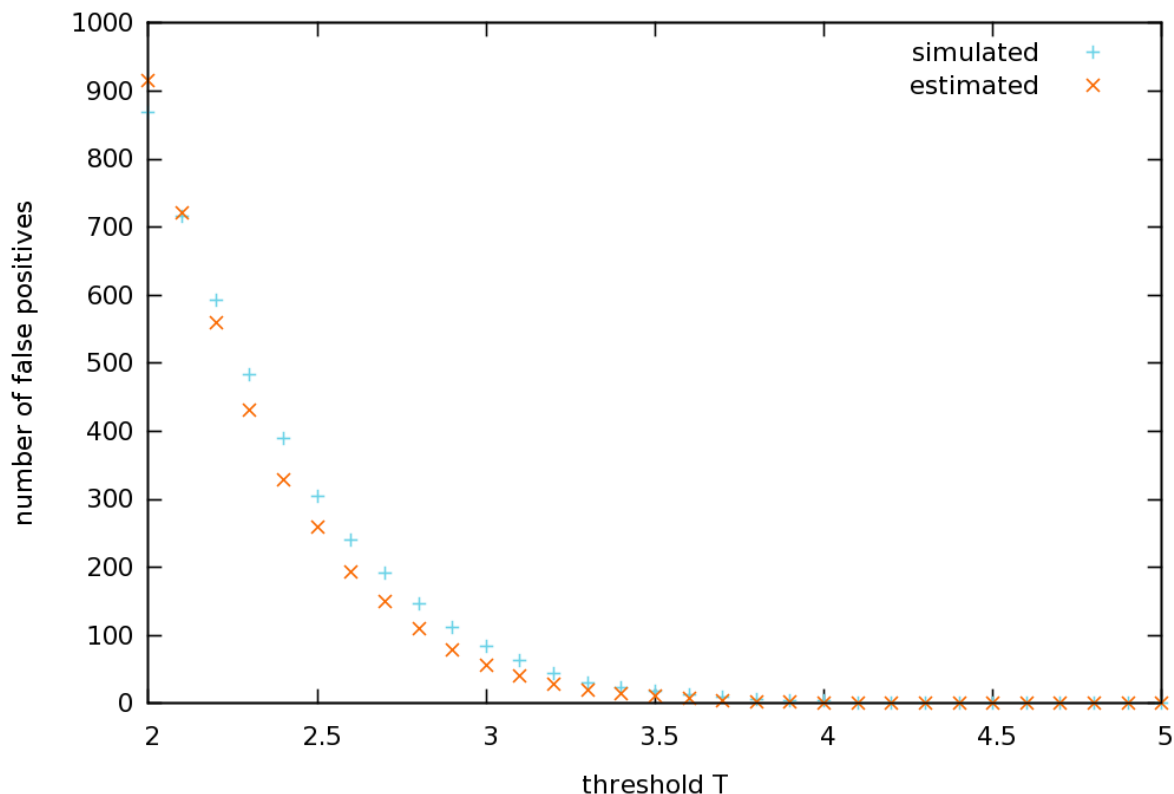


Figure 10: Number of false positives generated by the CE statistic for an assembly made from an ideal normal library. This graph uses  $\alpha = 0.5$ , and the two curves cross at  $T = 2.1$ . Choosing  $\alpha$  is a tradeoff between a good fit at small  $T$  and large  $T$ .

In practice insert libraries are not perfectly normal. While  $T = 3$  corresponds to events that occur with probability 0.003 for a normal distribution, this probability varies considerably for other insert library distributions. Thus, we created a simulated assembly using insert sizes from an actual insert library from an assembly of the rat genome (see figure above). This library has a weighted mean insert length of 2771 with a weighted standard deviation of 183. This simulated assembly consisted of five 10 Mbp contigs with average insert coverage of 50. We then computed the CE statistic for this simulated assembly. Note that this assembly is perfect with no real compression or expansion misassemblies, so any CE problem points found will be false positives. In the following figure we can see the average number of CE problem points found per contig for each

threshold  $T$ ,  $2.0 \leq T \leq 5.0$ .

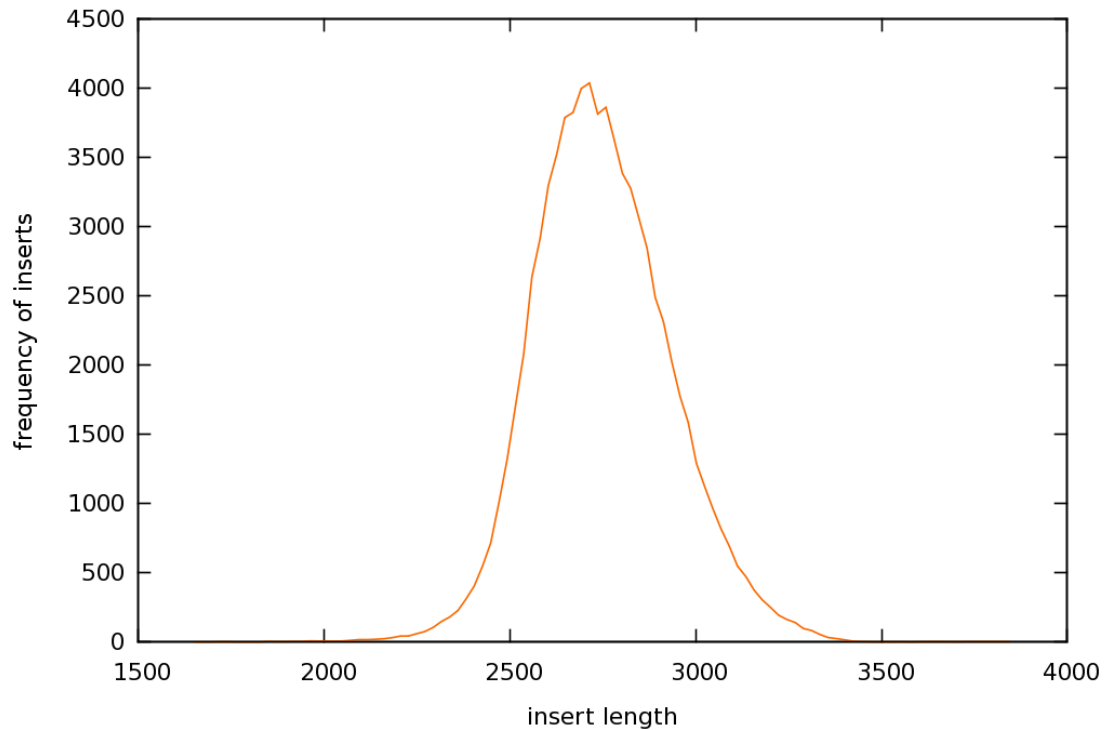


Figure 11: Histogram of the insert sizes for the 2771 library

It should be noted that in estimating the number of false CE problem points using the formula 8 above, we used a sampling distribution with sample size  $N = 50$ . In reality, insert coverage fluctuates along the assembly. For this particular simulation, insert coverage varied from 40 to 60 with an average coverage of 50. Varying  $N$  from 40 to 60 does not change  $p(N, T)$  significantly, and did not affect our estimates greatly. With the sample size varying from 40 to 60, the number of estimated false positives at  $T = 3.3$  varies from 40 to 45. The effect of varying coverage is more pronounced for poorer (further from normal) quality libraries.

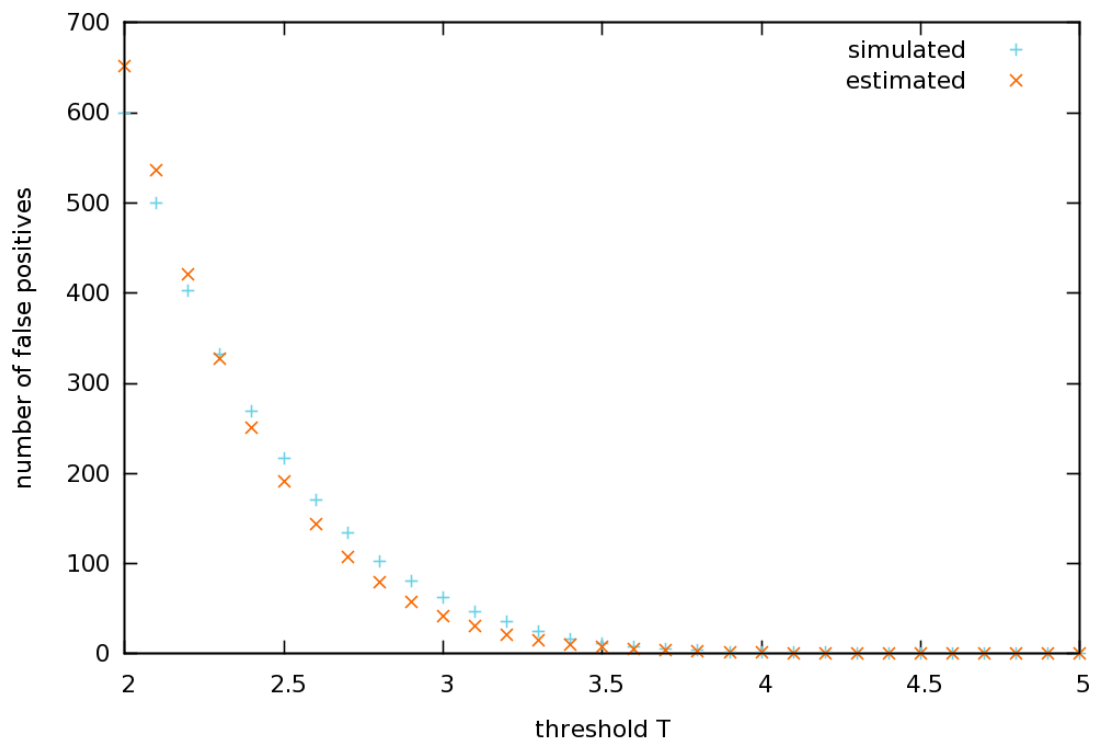


Figure 12: Plot of average number of CE problem points found in a perfect simulated assembly for the library with weighted mean 2771. The theoretical model (blue crosses) captures most of the variation in the number of false positives as a function of the threshold  $T$ . The model slightly overestimates the actual number of false positives (orange x's) when  $T < 2.3$  and underestimates it for  $T > 2.3$ .

**Detection of real CE misassemblies.** The previous experiments only give us information about the probability of seeing a false positive region. To find the sensitivity and false discovery rate of the CE statistic using the same library as above, we produced an assembly that was identical to the one described above with the exception that a CE misassembly of length 100-150 bp was inserted every 500,000 bp. We inserted an equal number of compression and expansion misassemblies. We would expect to be able to detect many of the 100 bp misassemblies for  $T = 3.8$  and below. We then computed the CE statistics for this assembly. Below is a plot of the sensitivity and false discovery rate of the CE statistic for different thresholds  $T$ . A good range of thresholds to choose from would be between 3.1 and 3.7.

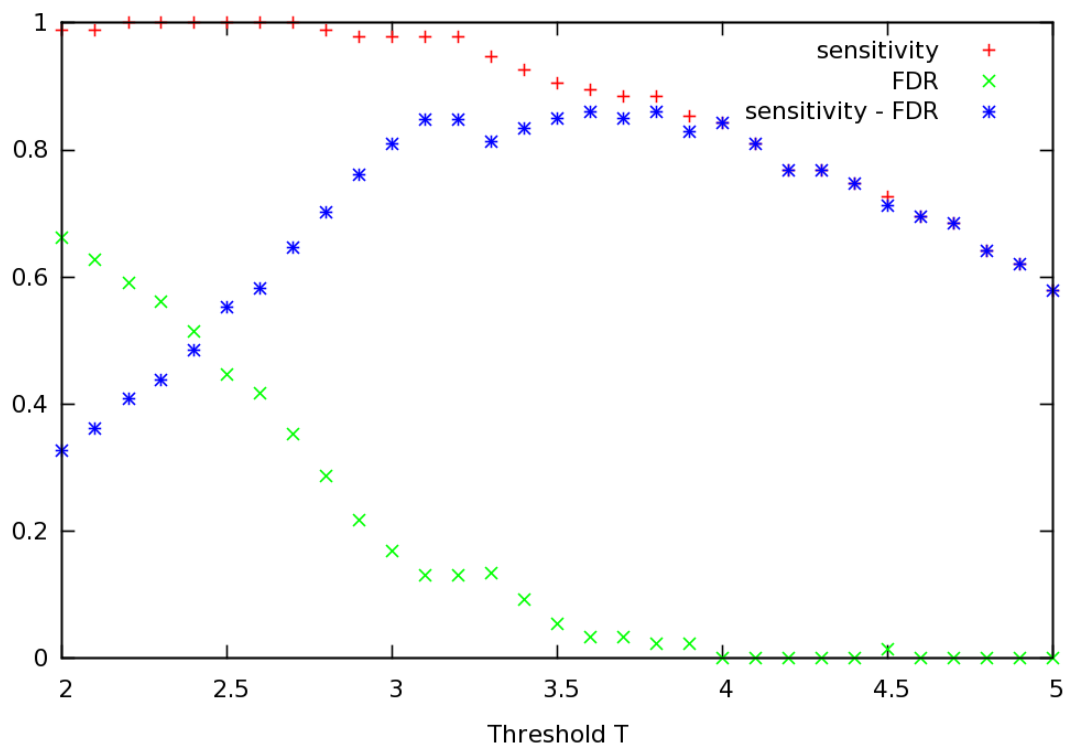


Figure 13: Plot of sensitivity (red crosses) and FDR (green x's) for different thresholds for the CE statistic using the library with weighted mean 2771. The blue stars represent the difference between the sensitivity and FDR which we would like to maximize. This can aid us in choosing a reasonable range of thresholds for the CE statistic. In this case, a good threshold would be  $3.2 < T < 4$ .

For some libraries, the insert coverage may not be high enough for the distribution of the sample mean to approach normal, especially if the libraries are poorly constructed. As an example, we chose another library from the rat assembly which was not as close to being normally distributed as the previous library. This library had a weighted mean of 2422 and a weighted standard deviation of 937. As we can see the standard deviation is almost forty percent of the mean. In practice, we often have libraries that lie between these two extremes.

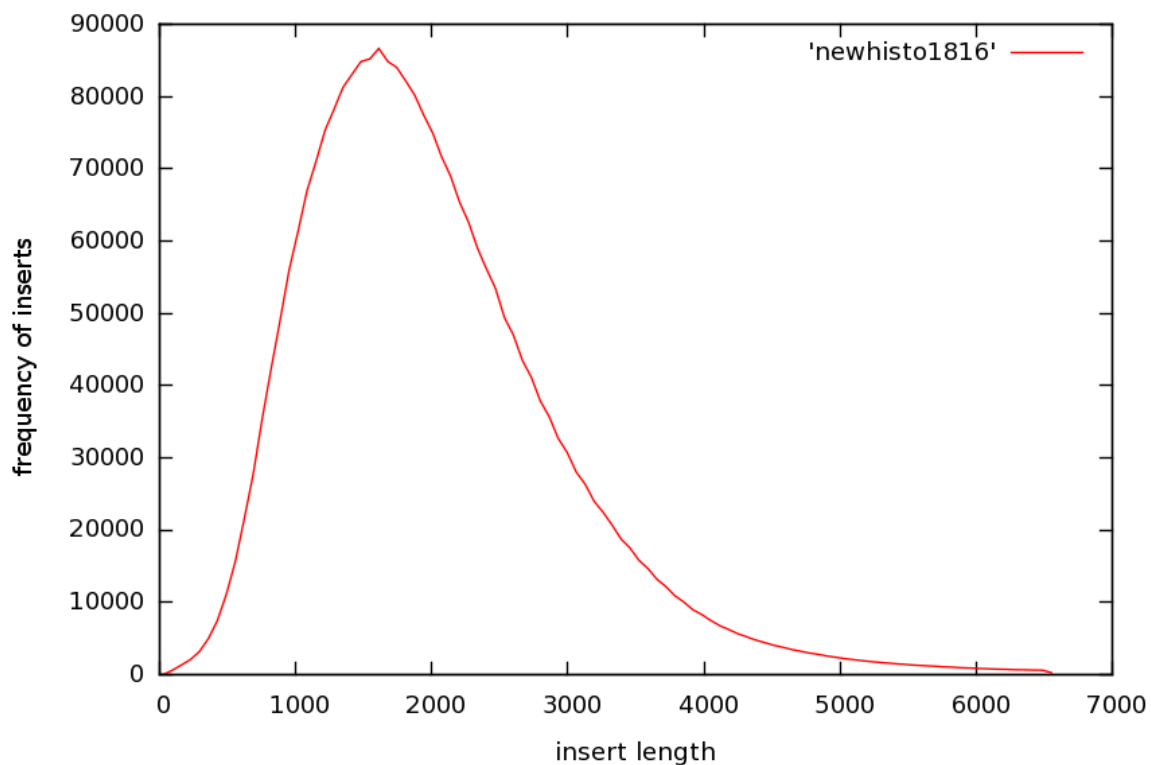


Figure 14: Histogram of insert sizes of the library with weighted mean 2422.

We again created an assembly as before including the alternating compressions and expansions. This time, however, since the standard deviation was so large, we increased the CE misassembly size to 500. The figure below shows a plot of the resulting sensitivity of the CE statistic using this library.

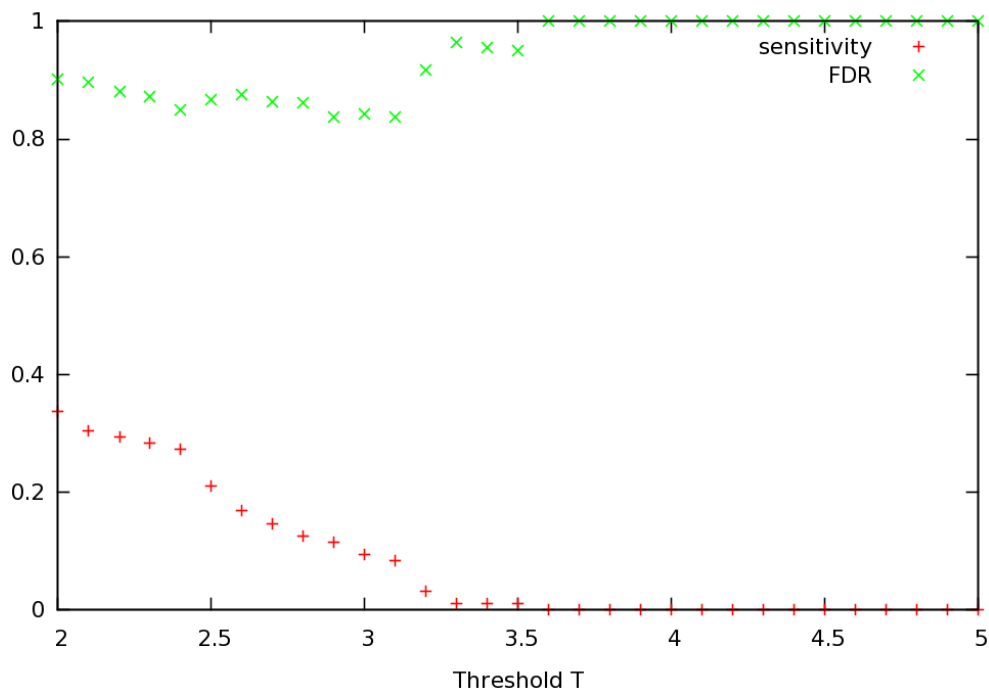


Figure 15: The sensitivity and FDR of the CE statistic for thresholds  $2 < T < 5$  for a simulated assembly. This assembly consisted of 5 10 Mbp contigs each with alternating 500 bp compressions and expansions every 500,000 bp. The assembly was made with a library with weighted mean 2422.

As we can see the sensitivity rate is lower than with the better distributed library.

If we increase the size of the misassemblies to 1500, the sensitivity and FDR improve:

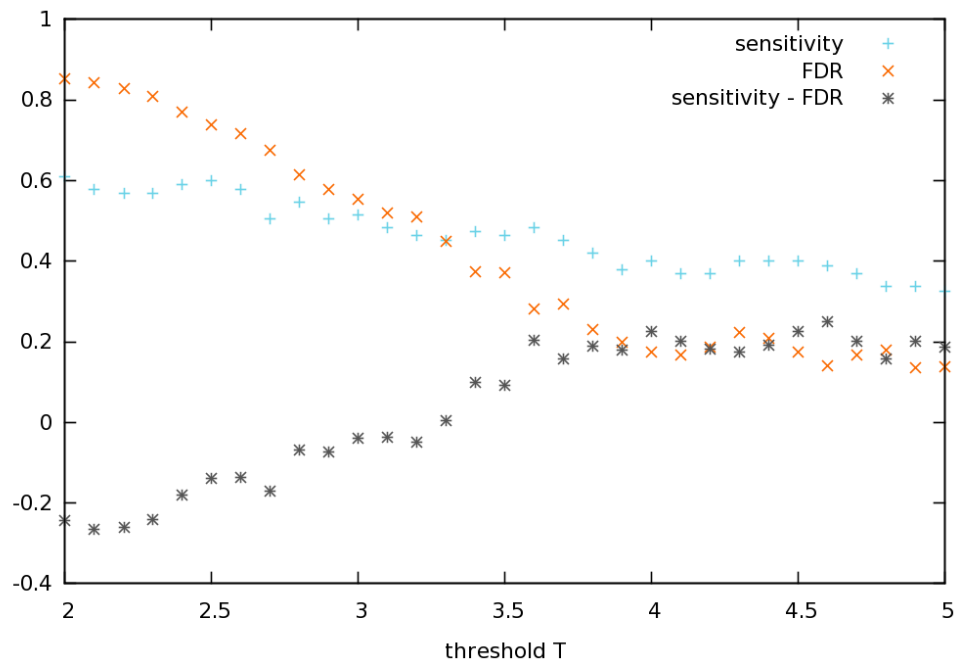


Figure 16: The sensitivity and FDR of the CE statistic for thresholds  $2 < T < 5$  for a simulated assembly. This assembly consisted of 5 10 Mbp contigs each with alternating 1500 bp compressions and expansions every 500,000 bp. The assembly was made with a library with weighted mean 2422.

Since we have seen that false positives can occur, we often need to use additional information to judge whether or not a CE problem point is a true misassembly. As we shall see, the Shooting Method can aid us in evaluating CE problem points.

## CHAPTER 3: THE SHOOTING METHOD and RESULTS

### Part 3.1: The Shooting Method

#### Section 3.1.1: The Shooting Algorithm

We will first discuss the general algorithm of our method for filling gaps between contigs and then describe how we adapt this method to fixing compression/expansion misassemblies.

**Initial Setup and Goal.** The existing information available for filling gaps is as follows. The positions of reads in contigs as well as an estimated mean and standard deviation of the size of the gap between two contigs are known. There is also a given pool of reads and a list of overlaps between these reads. We will discuss later how our dataset is chosen. For now, we will simply describe the algorithm. Given this dataset, our goal is to find all possible ways to fill a gap.

The first step in our method is to choose two reads, one from each contig on either side of the gap. Contigs are usually constructed by merging together overlapping reads until there are ambiguities as to what sequence comes next. These ambiguities can arise for three reasons. (1) There may be multiple ways to continue the contig due to a repeat or polymorphism in the genome; (2) there are no ways to continue the contig due to poor quality sequence at the end of the last read resulting in few to no overlaps; or (3) there are no ways to continue due to no read coverage of this region of the genome. For these reasons, we choose reads whose positions are a given threshold distance (usually one read length) from the end of a contig to ensure that we are not including reads that could contain erroneous or repetitive sequence.

In some cases, neighboring contigs may overlap producing a negative gap. In



these cases, we choose the reads to be at least one read length away from the overlapping regions in each contig. The goal is to find all possible ways to connect the contigs by replacing these overlapping regions.

We arbitrarily label one of these reads read 1 and the other read 2. Since an estimate of the gap size is known, we can infer a mean,  $\mu$ , and standard deviation,  $\sigma$ , of the length of the region between read 1 and read 2. Now, more specifically, our goal is to construct all possible assemblies of the region bounded by read 1 and read 2.

We can speak of a *consistent* set of “placed reads” (that is reads with positions) if every pair of reads in the set whose positions suggest they overlap do in fact overlap. A path is a collection of reads for which a consensus sequence exists; the consensus sequence should be connected (i.e. without gaps) and the collection of reads should include both read 1 and read 2. The Shooting Algorithm produces sets of read placements for each gap.

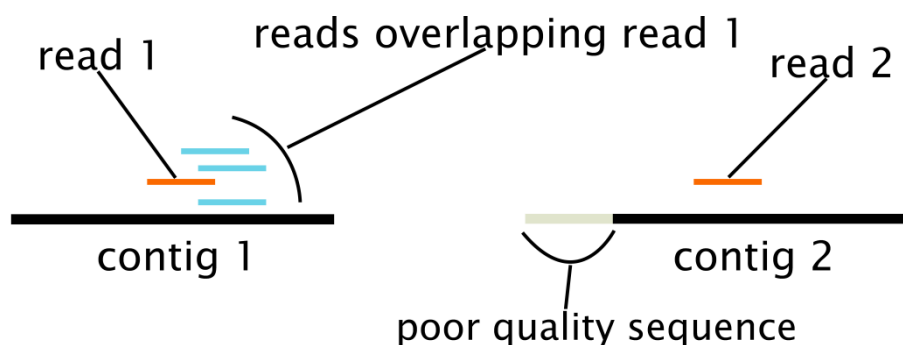


Figure 17: A schematic demonstrating the Shooting Algorithm. The two contigs, contig 1 and contig 2, flank a gap. The orange lines are the target reads chosen from each contig far enough away from the ends to avoid poor quality sequence (grey section of contig). We “shoot” from read 1 to read 2. The blue lines are reads which overlap read 1. They are the beginning of the wave front which will extend from read 1 to read 2.

**The Shooting Algorithm.** We start with read 1 and attempt to *shoot* for read 2 on

the other side of the gap building a *wave front* of reads. This terminology and approach is motivated by the standard shooting method for solving two-point boundary value problems for ordinary differential equations. Given read 1, we look for all overlaps in our list which are on the “right” side of read 1, the side closest to read 2 (See figure 18). We will refer to a read with a position as a placed read. We will refer to a read with a possible position as a read placement. We note these overlapping reads and their positions relative to read 1 as pairs  $(read_i, position_i)$ , for the  $i^{\text{th}}$  read in the list, and these pairs are added to the wave as the new wave front. Next, we look for all overlaps of these overlapping reads on their right sides and extend our wave again by adding the additional overlapping reads (and their positions) to the wave front. In this way we define the wave front to be the set of pairs whose right overlaps have not yet been checked.

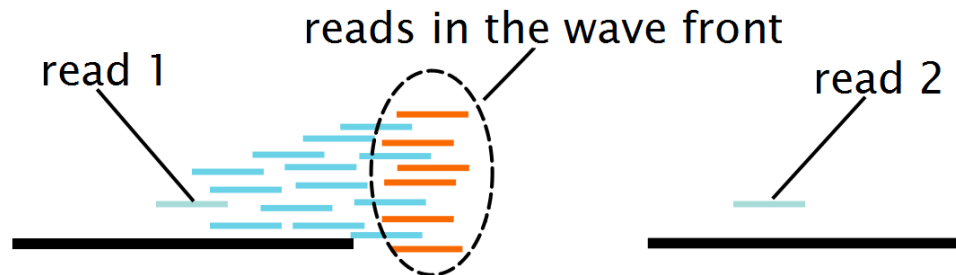


Figure 18: The orange reads represent the reads that make up the wave front as we iteratively build wave 1 from read 1 to read 2.

We set a maximum threshold,  $M$ , usually  $M = \mu + 3\sigma$ , such that if  $position_i > M$ , we do not add  $(read_i, position_i)$  to the wave front, and when there are no possible pairs that can be added to the wave front, this wave, *wave 1* (from read 1 to read 2), terminates. If read 2 never appears in the forward wave, we stop and fail to close this gap.

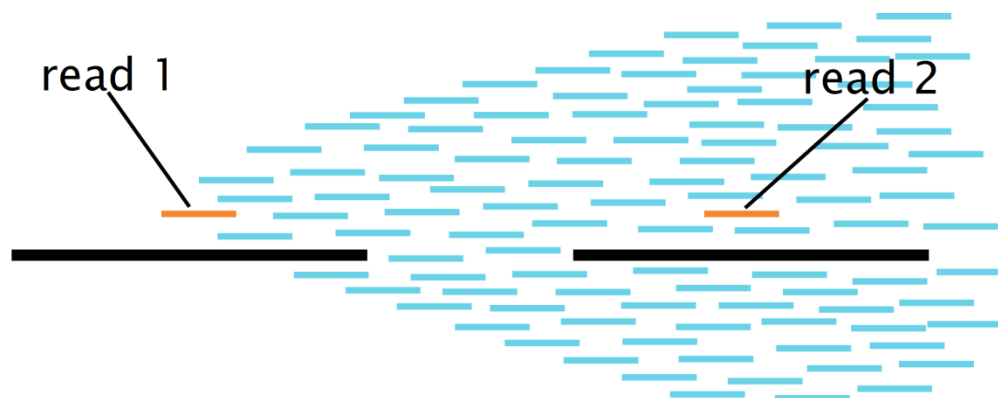


Figure 19: Wave 1 of the Shooting Algorithm.

If the wave 1 reaches read 2, we can trace back from read 2 to read 1 in a second wave, *wave 2*, using the exact same procedure as used in the forward direction, with one exception. This time, we use the pool consisting only of reads that are in the forward wave. The result is a smaller pool, *pool 2*, consisting of reads that can be reached from both read 1 and read 2. Tracing back eliminates any erroneous branches seen in the forward direction. Wave 2 includes all possible *(read, position)* pairs between read 1 and read 2 in this local region (see figure 21). Reads may occur in more than one position, indicating that there are multiple ways to assemble this region that result in different distances between read 1 and read 2. Sometimes, there are multiple ways to assemble a gap, only one of which is consistent with the estimated gap length and standard deviation. We will discuss this situation further in the next section.



Figure 20: The blue and orange segments denote reads in wave 1. The orange reads are the reads that are in both wave 1 and wave 2. Any path of reads between read 1 and read 2 must be built from the orange reads.

**Localizing the Dataset.** What we call the *global Shooting Method* uses the entire pool of reads available for the whole genome and all of the overlaps. However, there were two problems with this method. The first problem was that there were spurious overlaps in the set due to repetitive sequence. There were reads which overlapped each other, but which were from two different parts of the genome. This led to erroneous paths that would contain reads from portions of the genome that were not from our local region. For example, in testing our global Shooting Method on an assembly of *Brucella suis*, an organism for which the finished sequence is known, we encountered one case where the true sequence was 1346 bp long, but our algorithm also found a second assembly with length 1573 bp long. This spurious assembly was due to a section of sequence that had been inserted which didn't belong in this location. Aligning this section to the whole finished genome showed that this section was actually placed elsewhere.

The second problem was that sometimes our global set was missing overlaps between reads that belonged in our local region. This problem can occur because overlapping algorithms attempt to reduce the number of spurious overlaps in order to decrease computational time in assembly and reduce misassemblies. To this end, they may eliminate overlaps that occur in extremely repetitive portions of the genome. Therefore, these regions are often omitted in the draft assembly. Since these are precisely the regions we are trying to assemble with the Shooting Algorithm, we need to recover these overlaps.

There is additional information available to us that would allow us to recover the

missing overlaps. The types of additional information include mate pairs, BAC clone localization information, and targeted sequence. In our method, we mostly employ mate pair information to localize the set of reads. In sequencing projects using BACs (*bacterial artificial chromosomes*), large fragments of DNA on the order of 100 – 400 kbp are cloned by inserting them into a BAC and replicating them inside a bacterium. These fragments are then broken into smaller reads and sequenced as usual. However, it is known which reads come from which BACs, so one can localize to within a region 100 - 400 kb long. Finally, certain regions of the genome can be targeted for additional sequencing, so again it is known which reads come from these target areas.

We now create a smaller pool of reads from which to assemble the gap region. Our goal is to, as much as possible, include only the reads in the pool that are likely to belong to the region of the gap. Our pool consists of the following groups: (1) reads in the contigs which flank the gap; (2) mates of reads in group 1, which, according to the orientation and placement of the reads, would be expected to lie in the local region to be assembled; (3) reads which overlap the reads in groups 1 and 2 (given by the global set of overlaps); and (4) reads which overlap the reads in groups 1, 2, and 3. The smaller size of the pool allows us to compute more overlaps between these reads (which was computationally infeasible before) and to eliminate any spurious overlaps that would lead to false paths. This approach also speeds up the computation of our paths since our set of overlaps is much smaller than the global set.

**Producing Consensus Sequence.** Once we have produced a set of read placements for a possible assembly of the gap region, we must produce actual sequence. This final step can be done using many different tools. In our experiments (see Part III),

we used two algorithms to do this: (1) the make-consensus module from the AMOS assembler, and (2) the consensus module from the Celera Assembler. Both of these modules can take in the read placements produced by our Shooting Algorithm and produce a consensus sequence provided that our read placements are consistent. As we shall see in Part III, occasionally, our method produces inconsistent placements especially when haplotypes are present. Then, no consensus sequence is produced.

**The Shooting Algorithm in conjunction with the CE statistic.** As we described in Chapter 2, the CE statistic detects locations where it is plausible that there are compressions and expansions in a genome assembly. We have shown that the CE statistic can generate many false positives in identifying misassemblies. We can use the Shooting Algorithm to rule out these false positives, leaving a relative handful that can be investigated further, for example by doing additional targeted sequencing. In some cases, we can even correct these misassemblies.

To apply the Shooting Algorithm to a CE problem, we split the contig at the location of the CE problem point. The CE statistic provides an estimate of the size of the misassembly – it is  $M - \mu_w$  (the difference between the sample mean of insert lengths covering the location and the weighted library mean of insert lengths) as well as a standard deviation of this estimate. We use this value to estimate the size of the gap between the split contigs. We choose a read 1 and read 2 on either side of the suspected location of the CE problem region, again choosing these reads to be at least one read length away from the region. We then find all possible paths between these two reads as described above.

The Shooting Method can aid us in improving the sensitivity and false discovery

rate of the CE statistic. For example, if the Shooting Algorithm produces a unique path between read 1 and read 2 around a CE problem point, and the consensus sequence of this path matches that of the original assembly, then it is likely that this CE problem point is in fact not a real misassembly. We would consider all problem points for which there is no unique path matching the assembly to be a possible CE misassembly which should be further investigated.

We can also partially apply the Shooting Method to the set of mate pairs in the original data. We use the set of overlaps from the all of the reads in the original data set to walk from a read to its mate and back using the Shooting Method. We then record which inserts are found to have a single path between the two mates. We call such inserts *rigid*. Now we examine all CE problem points and see if there is at least one rigid insert which covers this location. We found that CE problem points that have no rigid inserts covering them are likely candidates for true CE misassemblies. If a CE problem point has at least one rigid insert covering it and the distance between the two reads is consistent with their placements in the assembly, then it is unlikely that this location is misassembled. For example, in our experiments, we computed the CE statistic for an assembly of *Listeria monocytogenes str. 4b F2365*. There were 6 CE problem points found, 5 expansions and 1 compression. We then used the Shooting Method to reassemble these 6 regions. We found only one way to assemble each region. The consensus sequences from these unique paths were identical to the regions in the assembly and when compared to the finished sequence. So in fact, these were all false positives.

If the Shooting Algorithm produces more than one path with differing lengths, it

is likely that the CE problem point in question is in fact a real CE misassembly. In the next section, we will see how we can evaluate which path is the most likely (both in the case of gaps and CE problem points).

### **Section 3.1.2: Multiple Paths**

The Shooting Algorithm sometimes produces more than one way to assemble a region. We describe below how we can sometimes rule out paths as being too long or too short. Multiple paths can result from a variety of reasons. Chief among them are polymorphisms and repeat regions in the genome. As we have discussed above, many organisms have chromosomes which come in pairs. Because of the way DNA is sequenced, sequence from both chromosomes is present in the read data. Therefore, it is occasionally possible to construct two different paths from these two different haplotypes. Both paths may be correct. In the case of repeat regions, the algorithm may produce paths that have different numbers of copies of repeats in them. Some paths could also contain parts of the genome not from this region. We would like to evaluate these different paths in order to find the one whose length is much more likely to occur than the others, if there is one.

The distribution of insert sizes spanning a gap can allow us to determine upper and lower bounds for the size of the gap and the likelihood of each path length. We compute these bounds by calculating the distribution of possible sizes for the gap. Each insert spanning the gap has its ends at known locations in the contigs. The length of the insert is unknown but comes from a known library distribution. Knowing how far the ends of the insert are from the gap implies a distribution of possible gap sizes. This shifted distribution has the same shape as the library distribution.



Let  $Q$  denote the set of inserts spanning a given gap, and let  $m$  denote the number of inserts in  $Q$ . Let  $\gamma$  be the mean size of the gap as estimated by the assembly program (or by the CE statistic in the case where the gap was introduced due to a CE problem point). We assume no information about the distribution of gap sizes is given. We consider each insert length to be an independent random variable from its library distribution. For each  $i$ , we shift its library distribution,  $P_i$ , by  $\gamma - \mu_i$ , so that it is a distribution of the implied gap sizes. We compute the probability distribution,  $P$ , of sizes of the gap as the distribution of a weighted average of these random variables representing the shifted insert lengths of inserts in  $Q$  (where the sum of the weights is of course 1). Choosing the following weighting ensures that we are minimizing the variance of  $P$  (Wild 2000): the weight  $w_i$  for the shifted length of the  $i$ th insert is

$$w_i = \frac{(1/\sigma_i^2)}{\sum_{j=1}^m (1/\sigma_j^2)}$$

We will denote the distributions of the weighted random variables (the shifted insert lengths) as  $P'_i$ , where the probability that an insert has length  $y$  is  $P'_i(y) = P_i(y/w_i)$ . Then  $P = P'_1 * \dots * P'_m$  where  $*$  is the convolution operator:

$$P'_i * P'_{i+1}(y) = \sum_k P'_i(k)P'_{i+1}(y - k).$$

We compute  $P$  by using the convolution algorithm in MATLAB.

Recall that the probability distributions  $P_i$  are computed from data and are far from smooth. We smooth  $P$  to eliminate high frequency oscillations, and we abuse terminology by calling the smoothed distribution  $P$  as well. We use this distribution  $P$  to evaluate the different paths given by our Shooting algorithm. We compute a confidence interval of  $P$  at a set level  $C$ , often 95%.

Our criteria depend on a likelihood ratio threshold  $R$  and are as follows: (1) The path length must be within the confidence interval  $C$  of our distribution. (2) If there are multiple paths, we consider the path length  $l^*$  which has the highest likelihood  $P(l^*)$ . Let  $l_i$  be the next most likely path length. If  $\frac{P(l^*)}{P(l_i)} > R$ , we accept the path of length  $l^*$  as the correct assembly. Otherwise, we simply report all possible assemblies satisfying criterion (1) and their likelihoods  $P(l)$ .

The only exception to these criteria is if we are considering a CE problem point (not a gap) for which the Shooting Method produces a unique path. If this assembly of this path is the same as the draft assembly in this location, we presume the problem point is a false positive and leave the assembly unchanged.

## **Part 3.2: Results and Discussion**

### **Section 3.2.1: Bacterial Assemblies**

In this section, we describe the experimental procedure we use to assess the Shooting Algorithm's ability to fix compression/expansion errors and fill gaps. We also discuss the results of these experiments. We test our method on a set of finished bacterial genomes.

Bacterial genomes are easier to assemble than those of organisms whose genomes are larger and for which, typically, repeats are a much larger fraction of the genome. Also, bacteria are also haploid organisms, and therefore polymorphisms are not present in their genomes. Recall that while such larger organisms would potentially be better test organisms, there are no large organisms whose genomes are finished and for which there is WGS read data available. While *Drosophila melanogaster* is essentially finished and was originally assembled using WGS data (Myers et al. 2000), significant parts of the

WGS data are not available. We chose to create draft assemblies of bacterial genomes that have been previously finished and for which WGS data is readily available. In our routines, we use the same reads (without prior error correction and trimming) as were used in creating the original draft genome, but not the reads used in finishing.

**Description of bacterial genomes.** We study 22 bacteria which were sequenced using Sanger techniques with sequence data (reads, mate, and insert library data) deposited in GenBank. These genome assemblies are considered *finished* according to the standards set by the International Human Genome Project. An assembly is considered finished if there are few to no gaps in the assembly and no more than 1 error per 10,000 bp (National Human Genome Research Institute 2009) .

Table 1 lists the bacteria that were used for this experiment and the sizes of their genomes.

| <b>Organism</b>                                  | <b>Genome size</b> |
|--|--------------------|
| <i>Listeria monocytogenes 4b F2365</i>           | 2.90 Mb            |
| <i>Wolbachia pipientis wMel</i>                  | 1.26 Mb            |
| <i>Bacillus anthracis Ames Ancestor</i>          | 5.23 Mb            |
| <i>Bacillus anthracis Ames</i>                   | 5.23 Mb            |
| <i>Burkholderia mallei ATCC:23344</i>            | 5.84 Mb            |
| <i>Brucella suis 1330</i>                        | 3.32 Mb            |
| <i>Streptococcus agalactiae 2603V/R</i>          | 2.16 Mb            |
| <i>Coxiella burnetii RSA 493</i>                 | 2.00 Mb            |
| <i>Campylobacter jejuni RM1221</i>               | 1.78 Mb            |
| <i>Chlamydophila caviae GPIC</i>                 | 1.17 Mb            |
| <i>Dehalococcoides ethenogenes 195</i>           | 1.47 Mb            |
| <i>Neorickettsia sennetsu Miyayama</i>           | 859 kb             |
| <i>Fibrobacter succinogenes S85</i>              | 3.84 Mb            |
| <i>Mycoplasma capricolum ATCC27343</i>           | 1.01 Mb            |
| <i>Prevotella intermedia 17</i>                  | 2.70 Mb            |
| <i>Pseudomonas syringae pv. tomato DC3000</i>    | 6.40 Mb            |
| <i>Staphylococcus aureus subsp. aureus COL</i>   | 2.81 Mb            |
| <i>Candidatus Koribacter versatilis Ellin345</i> | 5.65 Mb            |
| <i>Staphylococcus aureus subsp. aureus JH1</i>   | 2.91 Mb            |
| <i>Geobacter uraniumreducens Rf4</i>             | 5.14 Mb            |
| <i>Pelodictyon luteolum DSM273</i>               | 2.36 Mb            |

|   |         |
|---|---------|
| <i>Streptococcus thermophilus</i> LMD-9 . | 1.86 Mb |
|---|---------|

Table 1: Listing of bacterial genomes and their sizes which were used to evaluate both the CE statistic and the Shooting Method

**New draft assemblies.** We constructed draft assemblies of these bacteria using the Celera Assembler version 3 starting from overlaps computed by the UMD Overlapper (Roberts et al. 2004). We then computed CE statistic for each of these assemblies with a threshold  $T = 3.3$ , and applied the Shooting Algorithm to close gaps and validate and correct CE problem points. The same set of overlaps used in making the draft assemblies was used to create the pools of reads for the Shooting Algorithm. Recall that when we calculate the overlaps for the local pools of reads, we include all overlaps between reads. The global set of overlaps used by the Celera Assembler did not include overlaps from highly repetitive regions.

**Estimates of gap sizes.** As we saw in Part 3.1, our method depends on knowing an estimate of the size of the gap or CE misassembly. As we stated above, the CE statistic gives us an estimate of the size of the suspected expansion or compression. In the case of intrascaffold gaps, we use the estimate provided by the assembly program. In the Celera Assembler, the scaffolding module caps the negative gap size between two neighboring contigs at -20. For these gaps, we chose to use a mean gap size of 0 with a standard deviation equal to the standard deviation of the largest library. As we discuss in Section 3.2.2, the estimate of the gap size is a crucial piece of information contributing to the success of the Shooting Algorithm.

**Results.** The Celera Assembler, possibly the best assembly program currently in use, has been developed continuously for over 10 years. A major goal of assembly is to produce an assembly without gaps. It is not easy to improve on the quality of the

assembly produced by CA. Our achievement is that our techniques correctly close 23% of the CA gaps, where the read coverage was available, while making an error in 1% of the gaps.

The average N50 contig size over all the assemblies was 86,729 bp. The average gap size (for positive gaps) was 1352 bp. For negative gaps (where the contigs overlap), the average size of the overlap was 468 bp. For the CE problem points, the average suspected compression size was 1040 bp, and the average suspected expansion size was 866 bp. The average size of the local pools of reads used to build the paths for each gap and CE problem point was 519 reads.

To evaluate the sequences produced by the Shooting Algorithm, we align them to the finished sequence using the Nucmer module of the Mummer sequence alignment software (Kurtz et al. 2004). Those gaps and CE problem points where the new sequence matched the finished sequence were considered correctly closed. We considered a sequence to match if it aligned with 99% identity over 99% of the length of the sequence. We also aligned the entire set of reads to the finished sequence, and we only attempted to close 488 gaps of the 1810 total that had contiguous coverage by the original WGS reads. We were able to find at least one path for 117 of these gaps, and the path was unique for 86 of them.

The results are shown in the following tables. We set the likelihood ratio threshold  $R = 2$  when there were multiple paths, that is we chose the path that was at least twice as likely as any other path. We found that using this value seemed to maximize the number of correct paths (when aligned to finished sequence) minus the number of incorrect paths.

|                        | Correct | Incorrect |
|------------------------|---------|-----------|
| Unique path            | 86      | 5         |
| Multiple Paths         |         |           |
| ■ Correct Path Present | 8       | 10        |
| ■ Correct Path Absent  | N/A     | 8         |

Table 2: There were 1810 gaps in all of the bacterial draft assemblies. Of these, 488 had contiguous read coverage. Correct paths are those that match the finished sequence. For multiple paths, Correct Path Present means that the correct path was present in the set of paths generated by the Shooting Method. Correct Path Absent means none of the paths matched finished sequence. If the correct path was present, but using our criteria, we chose the wrong one, then this path is considered incorrect.

We see from Table 2 that if there is a unique path, it is the correct path in 95% of cases. However, when there are multiple paths, the method usually fails to identify the correct path, presumably because the initial estimate of the size of the gap is incorrect. In particular, the correct path may not lie within 3 standard deviations of the mean that the Celera Assembler provides (using the standard deviation that Celera provides). Most often, the Celera Assembler provides a mean that is too small. It would be possible to develop independent estimates of the gap sizes following the ideas (in Chapter 2) used in estimating the size of CE problems. Recall that the mean size of the gap or compression has to be corrected due to the tendency to have inserts that are mostly longer than average spanning the gap – due to the failure of shorter inserts to span the gap.

There were 153 CE problem points in all of the draft assemblies. 140 of them were identified as false positives because the Shooting Algorithm produced a single path which matched both the assembly and the finished sequence. In 7 of these cases, this was incorrect, and there was in fact a misassembly. The remaining 13 cases were identified as suspected CE misassemblies by our technique. These 13 cases can be investigated

further with targeted sequencing to determine the actual sequence. It is, nonetheless, interesting to see how often the Shooting Algorithm can correct the problems without further sequencing. In 5 of these cases, one of the paths that were produced was correct, and in 3 of these, we were able to choose the correct path according to our criteria.

|   |     |
|---|-----|
| Total CE problem points   | 153 |
| Unique paths matching assembly and finished sequence – false positives  | 133 |
| Unique path matching assembly, but not finished sequence (the assembly is wrong, but we did not identify these as errors) | 7   |
| Remaining points needing investigation  | 13  |
| No path   | 6   |
| Multiple paths  |     |
| ■ Correct path present and not chosen   | 2   |
| ■ Correct path present and chosen   | 3   |
| ■ No correct path   | 2   |

Table 3: Results of the Shooting Algorithm applied to CE problems in the bacterial assemblies. If there was a unique path matching the finished sequence and the assembly, we conclude that the CE problem point was not a CE misassembly. In all other cases, the draft assembly contained a misassembly. Rows 5 and 6 categorize the 13 CE problems that are suspected to be misassemblies by our method.

There were 8 CE misassemblies in the entire collection of bacterial assemblies. Of these, the CE statistic identified 6 as CE problem points. After using the Shooting Algorithm to rule out potential false positives, there were 5 remaining cases that were correctly identified as CE misassemblies. We were able to choose the correct path in 3 of these cases.

### Section 3.2.2: Limitations of the Method

As with any computational method, our algorithm is only as good as the data we give it. Sequence data is notoriously inconsistent, so every dataset we encounter has different challenges. We attempted to make a robust method that could be modified to

work effectively on most genome assemblies, but of course there are limitations. The biggest limitation we have is that we are dependent on the estimated size (mean and standard deviation) of the gaps we are trying to close (or the size of the compression/expansion) in the draft assembly. A related problem occurs when we do not have an appropriate library that could fill the gap. And finally, sometimes we cannot tell how many copies of a short repeat sequence should be placed in a gap.

**Poor estimate of the gap size.** Since we allow the assembler to estimate the gap size, we often do not properly assemble the region if the gap size estimate is incorrect. In the case of the Celera Assembler, it is difficult to evaluate the validity of its estimates of the mean and standard deviation of the gaps because in its evaluation, it chooses some mate pairs as correct and others as incorrect and bases its estimate on the pairs it has declared correct. We do not know how it makes this determination of “correctness.”

**Example – underestimated gap size.** This example illustrates two independent phenomena. First, if the gap size estimate is too short, then the Shooting Algorithm could stop before reaching read 2, or if it finds multiple paths (of different lengths), it can choose the wrong path, or all paths generated can be too short. This example is of note because it contained 26 copies of a *tandem repeat*, that is, situations where copies of the repeat unit are adjacent to each other with no other sequence in between. As we shall see later on, in situations with such a high copy number (where the collection of repeats is longer than any read), it is impossible to determine the number of copies without additional sequencing together with careful analysis.

The CE statistic detected a compression in the assembly of the bacteria *Pelodictyon luteolum* DSM 273 in this location with an estimated gap size of 3552 bp



long. However, in the finished sequence, this gap was actually 7652 bp. The library which detected this misassembly had a weighted mean of 6794 bp and was the longest library available for this assembly. So it is unlikely that we could detect the true length of the compression. The inserts which span this location must be longer than average in order to span a compression of this size. In principle, if we ignore mate pair data, any number of copies of the repeat would be possible. In the finished sequence there were 26 copies of a repeat unit with length 363 bp. Our threshold for how far we should extend our paths was too short to recover all 26 copies. The longest path the Shooting Algorithm produced had only 22 copies of the repeat. In section 3.1.2, we described how we compute likelihood estimates for comparing different path lengths. Since our estimate of the gap size was too short, the range of acceptable paths for correcting this compression contained between 12 and 16 copies of the repeat.

**Example -- overestimated gap size.** Conversely, if the estimated gap size is too large, we will usually still recover the correct path. However, if there are multiple paths, we may not choose the correct one when deciding whether the path is likely to occur. One example of this is a case where there was a true expansion in the draft assembly; two neighboring contigs overlapped by 789 bases when aligned to finished sequence. However, the Celera Assembler reported the gap size between the contigs to be 1 bp. Our shooting method found two ways to assemble this region. The correct path was, of course, deemed to be unlikely since it was much shorter than the expected length of the path. The other path found was actually too long; it contained two short repeats on the order of 200 bp, as well as some extra sequence in between. The algorithm may not have found this other assembly if the gap size had been correctly estimated. It would not have

overshot the correct path. These examples show that our method is quite dependent on accurate scaffolding of contigs and a good estimate of gap size by the CE statistic.

**Poor collection of library sizes.** Having a variety of library sizes is important to our method for two reasons. First, when our software creates the initial pool of reads from which to assemble a local region, it needs reads whose mates are expected to lie in the gap of the draft assembly (or the disputed region in the case of an expansion). Second, when we are trying to determine which path is more likely to occur, we are relying on there being inserts that span our gap in order to determine the distribution representing the possible gap sizes. Because insert sizes affect gap closing in two different ways, we would like library sizes that are well distributed (not to be confused with the distribution of insert sizes within a library). Shorter libraries tend to have smaller standard deviations. We need library sizes which are short enough that the mates of reads which flank a gap can plausibly lie in the gap, but we also need libraries that are long enough for the inserts with mates in flanking contigs to fill the gap. Sometimes, the mean of the library distribution was around the size of a gap, which means only longer inserts were actually spanning the gap, which skewed the joint distribution of the gap size. Because of this, the way in which the collection of insert libraries is constructed is crucial to the success of our method.

**Poor distribution of inserts lengths within a library.** Another problem we encountered was libraries with poor distributions, nearly bimodal with large standard deviations. Ideally the distributions should be nearly normally distributed with a single peak. See figure below which appears to have two libraries merged into one. Bimodal distributions can arise when the laboratory producing the inserts labels two libraries with

different insert sizes as being a single library.

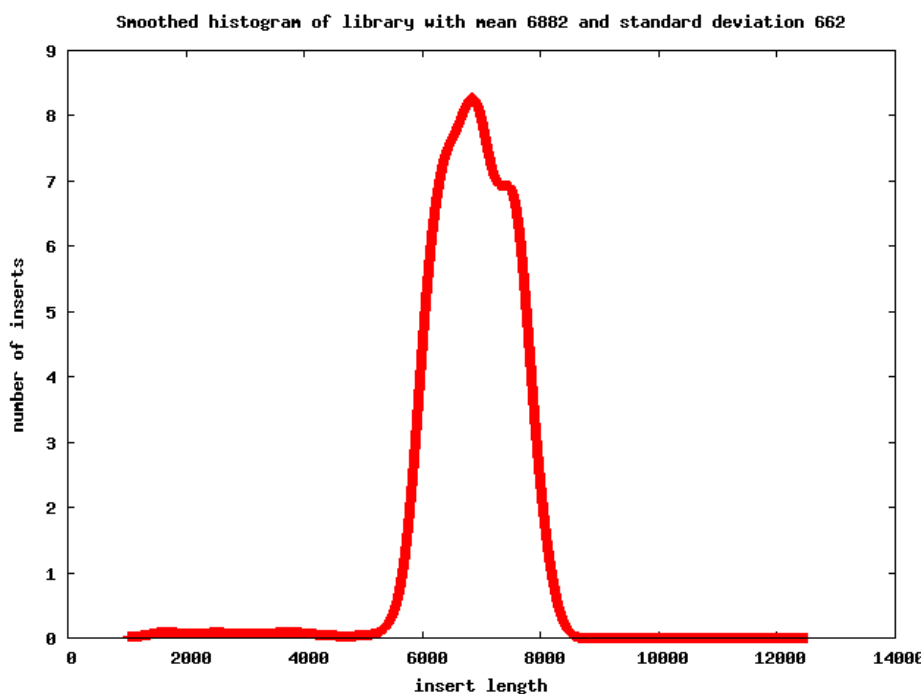


Figure 21: A smoothed histogram of a library from an assembly of *Acidobacterium bacteria Ellin345* with  $\mu = 6882$  and  $\sigma = 662$ . Note the double peak.

**Trouble estimating the copy number of repeats.** The third limitation of our algorithm is that sometimes it cannot choose between different paths when there are short repeats in the sequence. In one instance, in an assembly of the bacterium *Burholderia mallei*, there was a gap in the draft. The finished sequence contained the pattern  $ARBRC$ , where  $A$ ,  $B$ , and  $C$  are unique sequences in this region, and  $R$  is a repeat unit of length 135 bp. The Shooting Algorithm was able to assemble the correct sequence, but it also produced some longer paths which contained additional copies of the sequence  $RB$ . If  $RB$  is longer than a read, the available read overlaps will not be able to rule out any of the patterns  $A(RB)^n RC$ , for any  $n = 1, 2, 3, \dots$ . The set of overlaps cannot rule out  $RB$  being a repeat though in the finished sequence it is not. Furthermore,  $ARC$  is a path that the

algorithm will produce, and it currently can only be ruled out by being too short. Since the repeat unit  $RB$  is so short, it was difficult to tell which path was correct using our criterion for choosing among multiple paths. While the chief criterion we have for differentiating between different paths is path length, we can also examine the placements of the reads in the paths along with the implied placements of their mates to assess if one path fits better in this region than another. However, in all cases we checked in this experiment, this was not helpful in choosing between the paths which could, in cases of short repeats like these, differ by only one read. This was not enough of a difference.

In principle, read coverage of the sequence  $B$  might allow an algorithm to accurately estimate the number of copies of  $B$ , that is, the value of  $n$ . We did not build read coverage into the Shooting Algorithm because our approach is to produce local assemblies of gap regions using relatively small pools of reads. To compute the read coverage, we would need to align all reads from the entire global data set to this region, which is outside the scope of the local assembly. Our local methods are likely to miss some of the reads that could be placed in the gap, reads that are not necessary to create the set of paths. The existence of a gap is often connected with the presence of a copy of a repeat, which might occur many times in the genome. In such a case, a huge number of reads might be consistent with the gap sequence, and we chose not to create a pool involving all such reads (see Section 2.1.1).

It should be noted that this last difficulty in determining the copy number of repeats affects only a small fraction of gaps and CE problem points. Better insert libraries (with smaller deviations) would solve some of our problems.

**An alternative not pursued.** One approach to estimating a gap size uses the number of inserts (from a library) spanning the gap and infers from this the size of the gap. Choose a gap size and an insert coverage of the genome. Create random samples of inserts and determine by experiment the expected number that span the gap and its standard deviation. The expected number would decrease as the gap size increases. From this experiment, a likely range of gap sizes could be estimated.

### **Section 3.2.3: *Bos taurus***

**The Assemblies of *Bos taurus*.** The Baylor College of Medicine assembled the cow genome and published its results in *Science* (The Bovine Genome Sequencing and Analysis Consortium et al. 2009). Simultaneously, our group published our own assembly (Zimin et al. 2009) using the same data as the Baylor assembly. Our group reassembled the cow genome using the Celera Assembler version 4.2 and this data which was obtained from the NCBI trace archive. This resulted in a draft assembly which had a total of 81,556 contigs mapped to the chromosomes in 11,082 scaffolds, many of which were single contig scaffolds. There were 59,983 intra-scaffold gaps. My objective in this project was to use the Shooting algorithm to close as many of these gaps as possible.

**Using repetitive overlaps.** As described earlier, genome assemblers are designed to ignore the overlaps of reads that cover highly repetitive sections of the genome, because a majority of such overlaps would be spurious and thus using them will be problematic for the assembly. The strength of our Shooting Method is that it constructs a local assembly using smaller pools of reads. As we mentioned in the methods section, the reads in these pools are chosen using mate pair and overlap information. We can then compute all overlaps for each pool of reads and that may include some overlaps that were

originally discarded by the assembler. This provides an advantage over the assembler and will allow us to close some gaps that the Celera assembler was unable to.

**Gap closing statistics.** We attempted to close all gaps whose size was less than 30,000 bp as estimated by the Celera Assembler or whose size was unknown. The number of gaps meeting this criterion was 58,386. Using the Shooting Method, we were able to find paths spanning a total of 12,374 gaps, of which 10,076 had a unique path length (though not necessarily a unique path). We remark that in our investigations of bacteria, every gap for which we produced a unique length in fact had a unique path with a consensus sequence. The average size (using Celera's estimate) of the gaps that we attempted to close was 3,613 bp. This average ignores the gaps for which we had no estimate. The average size of the gaps we were able to close was 2,704 bp. The average size of the pool of reads used to assemble the local regions was 210.

We were able to produce consensus sequence for 9,863 paths using the make-consensus program that is part of the AMOS package (cite). The consensus algorithm failed for some gaps with unique path length, because the only characteristic that we use to define a path is the path length. If there are two paths of the same length, which may occur due to haplotype differences, all reads in those paths would be reported together. This will cause the consensus program to fail.

**Assembly improvement.** We used these gap sequences (i.e. consensus sequence from a unique path) to join together the contigs on either side of the closed gap. In some instances three or more contigs were merged together with the gap sequences into a single contig. The additional sequence generated from the Shooting Method totaled 8.34 Mbp. In total, we were able to merge 16,814 contigs into 7,945 contigs. Of these 7,945

merged contigs, 5,068 were placed on the chromosomes. The total number of placed contigs decreased to 75,880 contigs, a decrease of 7%.

A metric used to evaluate assemblies is the *N50 contig size*. The N50 contig size is the size of the largest contig such that it plus all larger contigs together contain at least 50 percent of the assembly. For the original assembly, the N50 contig size was 87,934 bp. After gap closing, the N50 contig size increased to 94,027 bp, an increase of 7%.

**Removing redundant contigs.** Closing gaps not only generated additional sequence for the assembly, but it also helped us place that sequence in the correct positions. Draft genomes (usually the word “draft” is omitted) can have contigs that are misplaced: the mate pair information allows multiple orderings of contigs, and the assembler or the assembly team chooses one of these orderings. Alternatively, the contig may be placed in its own scaffold. In some cases, repeat regions in the genome caused sequence that should be in a gap to be reported in a separate single-contig scaffold. In these cases such sequence was placed incorrectly around the gap on the chromosome by the chromosome-building algorithm (Zimin et al., 2009).

Haplotype sequence was dealt with in a similar way. The original DNA for this assembly came from two *B. taurus* individuals, a bull and his daughter. This additional heterozygosity increased the number of haplotype differences present in the data which complicated the assembly process, and was a potential cause for some of the gaps. The Celera Assembler and the chromosome-building algorithm place contigs so that they do not overlap. Therefore contigs which should lie on top of each other (based on mate pairs), but cannot be merged into a single contig, are sometimes placed next to each other in the assembly instead. The fact that the two contigs were not merged by the assembly

program suggests either the two contigs came from different homologous chromosomes--that is they are different haplotypes--or there is bad sequence on the end of a contig, very low quality sequence from the end of a read.

After doing gap closing and placing the new merged contigs on the chromosomes, some of this misplaced sequence became redundant to the assembly. To remove this redundant sequence, all of the new merged contigs were aligned to contigs placed nearby (see figure 23). In some cases, a neighboring contig aligned well to the new gap sequence (see figure 23). In some cases, a neighboring contig aligned well to the new gap sequence with greater than 97 percent identity for greater than 90 percent of the length of the contig. These neighboring contigs were deemed to be redundant and were removed from the assembly. 1,439 contigs were removed from the chromosome assembly in this fashion. However, if the redundant contig is not identical to the sequence in the gap, we make a note of this sequence as being a possible haplotype variant.

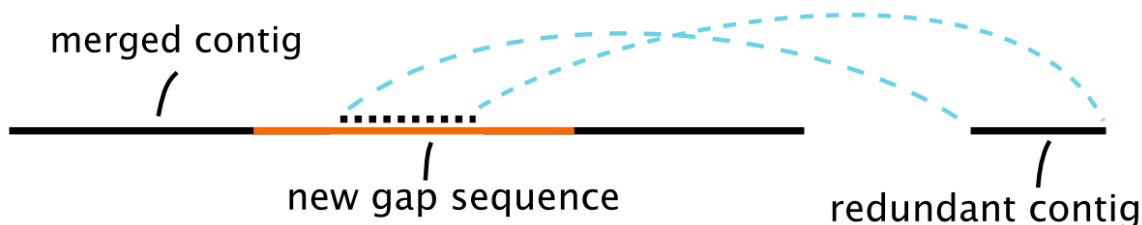


Figure 22: A schematic showing a merged contig containing two “original” contigs (black segments), and new gap sequence produced by the Shooting Algorithm (orange segment) as well as a now redundant contig placed next to the merged contig on the chromosome. The redundant contig’s sequence matches that of the gap sequence, and so it is removed from the assembly. This contig was probably erroneously placed next to the other contigs instead of being placed between them in the original assembly. The sequence is now correctly placed.

**Difficulties in closing gaps: poor collection of library sizes.** As we discussed in Section 3.2.2, library sizes heavily impact the success of our algorithm. In this assembly project, the sequencing center chose to make a variety of libraries of mean size about 3000-6500 Kbp (perhaps motivated by their BAC-by-BAC assembly process). They also



had a library of mean size 150 Kbp (which was useful in scaffolding) with a large standard deviation. Large libraries tend to have large standard deviations of their insert lengths. The library choice makes it difficult or impossible to detect CE misassemblies of size greater than 6Kbp. Similarly, a gap of 10 Kbp cannot be accurately estimated. Also, 6 Kbp would not give us enough mate pairs to build a local pool of reads from which to fill the gap, as the mates of the reads in the flanking contigs would not reach the middle of the gap. The best datasets are ones in which there are libraries which vary in size.

**Difficulties in closing gaps: waves that are too complex.** The *B. taurus* genome was much larger and more repetitive than the particular bacterial genomes discussed in section 3.2.1. Therefore, it was more challenging to assemble, and there were problems which we did not face in our previous experiments. We could not close some gaps because the gaps were long. Also, the estimates of the gap size for some of the gaps could have been incorrect. In running our Shooting Method, we limited the number of overlapping reads in each wave to 10,000. We chose the bound of 10,000 to limit run time and because we rarely found that using larger upper bounds yielded additional gap closings (move to methods section). For very long gaps, this may not have been enough to close the gap.

**Difficulties in closing gaps: contigs that are too short.** Another reason we had difficulty in gap closing in the cow genome was that some of the contigs flanking a gap were quite short. Recall that we create our local pools of reads based on mate pair and overlap information of the reads in the flanking contigs. If there were not very many reads in these contigs, then we would not have enough reads in the pool to close the gap.

Finally, the placement of the contigs by the scaffolding algorithm in the Celera Assembler and the chromosome-building algorithm may have been incorrect. It may have in fact been impossible to find any paths between some of the contigs in our experiment as their sequences may not actually lie near each other in the genome.

From our studies with the bacteria in the previous section, we found it was difficult to distinguish between multiple paths produced by the Shooting Method for a single gap unless there was an improvement in estimates of gap sizes. At the time of the assembly of *B. taurus*, no such improvement had yet been made, so this portion of our algorithm was not used here.

Potential application of the Shooting method: Using the Shooting method on mate pairs where one mate is in a contig, and the implied placement of the other is in a gap. Hence, we would determine the sequence between the two mates, and thereby we would extend the contig. This would allow us to detect bad sequence on the ends of contigs, which in turn might allow more gaps to be closed.

### **Section 3.2.4: An Assembly of Simulated Short Read Data from Human**

#### **Chromosome 1**

In this section, we describe an application of the CE statistic and Shooting Algorithm to an assembly of human chromosome 1 produced by Celera Assembler 5.4 from simulated or *faux* short read data. The reason we chose to assemble faux data is because there are no finished eukaryotic genomes for which enough short read data to produce an assembly is publicly available. The CE statistic and Shooting Algorithm do not depend on the size of the individual reads. Therefore, they are directly applicable to

assemblies produced from short read data. The purpose of this experiment is to illustrate this fact.

**Description of the faux data set.** In this experiment we used the most recent publicly available sequence of human chromosome 1 by the Human Genome Project Consortium (Genome Reference Consortium 2010). This sequence contains some gaps. For the purposes of this experiment, we simply removed these gaps and put the sequence together into a single contiguous sequence. We simulated 20X coverage by 100 bp reads. We did not introduce any errors in read sequences. Read positions were chosen from a uniform distribution along the sequence. 10X coverage was by unmated reads and the remaining 10X coverage was by mate pairs of reads in 3 different libraries (see Table 1). These libraries were chosen to simulate the actual libraries that can be produced by Illumina sequencing technology (cite).

| $\mu_w =$<br>Weighted<br>Mean<br>Insert Size | $\sigma_w =$<br>Weighted<br>Standard<br>Deviation | Average<br>insert<br>coverage | N =<br>Average<br>spanning<br>insert<br>coverage<br>(excludes<br>reads) | Compressions<br>that can be<br>detected at T =<br>3.5 (heuristic<br>estimate) | Sizes of<br>compressions<br>found<br>excluding<br>5% smallest<br>and largest |
|--|---|-------------------------------|---|---|--|
| 10,236 bp                                    | 1481  | 48                            | 47  | ~800 -10,000 bp   | 697-1556 bp  |
| 2044 bp                                      | 297   | 20                            | 18  | ~250 -2000 bp   | 223-654 bp   |
| 406 bp                                       | 50  | 14.5                          | 7.4   | ~50-200 bp  | 58-147 bp  |

Table 2: The size of compressions that can be detected. The *spanning size* of an insert is the size of the insert minus 200 bp, the total length of the reads at the ends of the insert. This size is used in computing the *average spanning insert coverage*. Recall that we estimate the minimum size of the compression that can be detected from the formula  $(T\sigma_w)/\sqrt{N}$ . This is an underestimate since the spanning insert coverage decreases at the site of a compression. The upper bound is the spanning size of the insert library. Because of the fluctuations in insert size, occasional compressions outside of the range may be detected.

We then used the Celera Assembler version 5.4 to assemble this faux data. The assembly statistics are listed in Table 2.

| Scaffolds | Contigs | N50 contig size | Total assembly size                    |
|-----------|---------|-----------------|--|
| 105       | 3971    | 213 Kbp         | 217Mbp (original chromosome is 225Mbp) |

Table 3: Statistics of faux assembly of human chromosome 1. Recall that the N50 contig size is the largest contig such that the sum of the number of bases in all larger contigs comprise 50% of the assembly. The original chromosome is the finished sequence.

**CE misassemblies.** We aligned the assembly to the original finished sequence and identified 482 compression misassemblies and 31 expansion misassemblies. The reason why the number of compressions was larger than the number of expansions is that assemblers have difficulty assembling repeat regions, which are the most frequent cause of compression misassemblies. Recall that for tandem repeats where each copy of the repeat is longer than the read length, the Celera Assembler can only estimate the number of appears to err on the side of fewer copies of the repeat region rather than extra copies.

We computed the CE statistic for this assembly. The assembly has so few expansions that the false positives greatly outnumber the true positives for each threshold  $T$ . The following table shows the sensitivity and false discovery rates for the CE statistic in detecting expansions at various thresholds  $T$ :

| Threshold $T$ | # true positives out of a total of 31 | # false positives | Sensitivity | FDR   |
|---------------|---------------------------------------|-------------------|-------------|-------|
| 2.1           | 26                                    | 60042             | 0.839       | 1     |
| 2.3           | 20                                    | 42517             | 0.645       | 1     |
| 2.5           | 17                                    | 27585             | 0.548       | 0.999 |
| 2.7           | 16                                    | 16920             | 0.516       | 0.999 |
| 2.9           | 14                                    | 10098             | 0.452       | 0.999 |
| 3.1           | 13                                    | 5595              | 0.419       | 0.998 |
| 3.3           | 10                                    | 2890              | 0.323       | 0.997 |
| 3.5           | 10                                    | 1461              | 0.323       | 0.993 |
| 3.7           | 8                                     | 746               | 0.258       | 0.989 |
| 3.9           | 9                                     | 350               | 0.29        | 0.975 |
| 4.1           | 8                                     | 189               | 0.258       | 0.959 |

|     |   |     |       |       |
|-----|---|-----|-------|-------|
| 4.3 | 7 | 100 | 0.226 | 0.935 |
| 4.5 | 7 | 60  | 0.226 | 0.896 |
| 4.7 | 7 | 36  | 0.226 | 0.837 |
| 4.9 | 7 | 29  | 0.226 | 0.806 |
| 5.1 | 7 | 24  | 0.226 | 0.774 |
| 5.3 | 6 | 23  | 0.194 | 0.793 |
| 5.5 | 6 | 19  | 0.194 | 0.76  |
| 5.7 | 5 | 18  | 0.161 | 0.783 |
| 5.9 | 5 | 15  | 0.161 | 0.75  |

Table 4: Sensitivity and false discovery rate (FDR) of the CE statistic in detecting expansions for various thresholds  $T$ .

Below, you can see a graph of the true compressions that were detected by the CE statistic with  $T = 3.5$  according to which library was used in the detection:

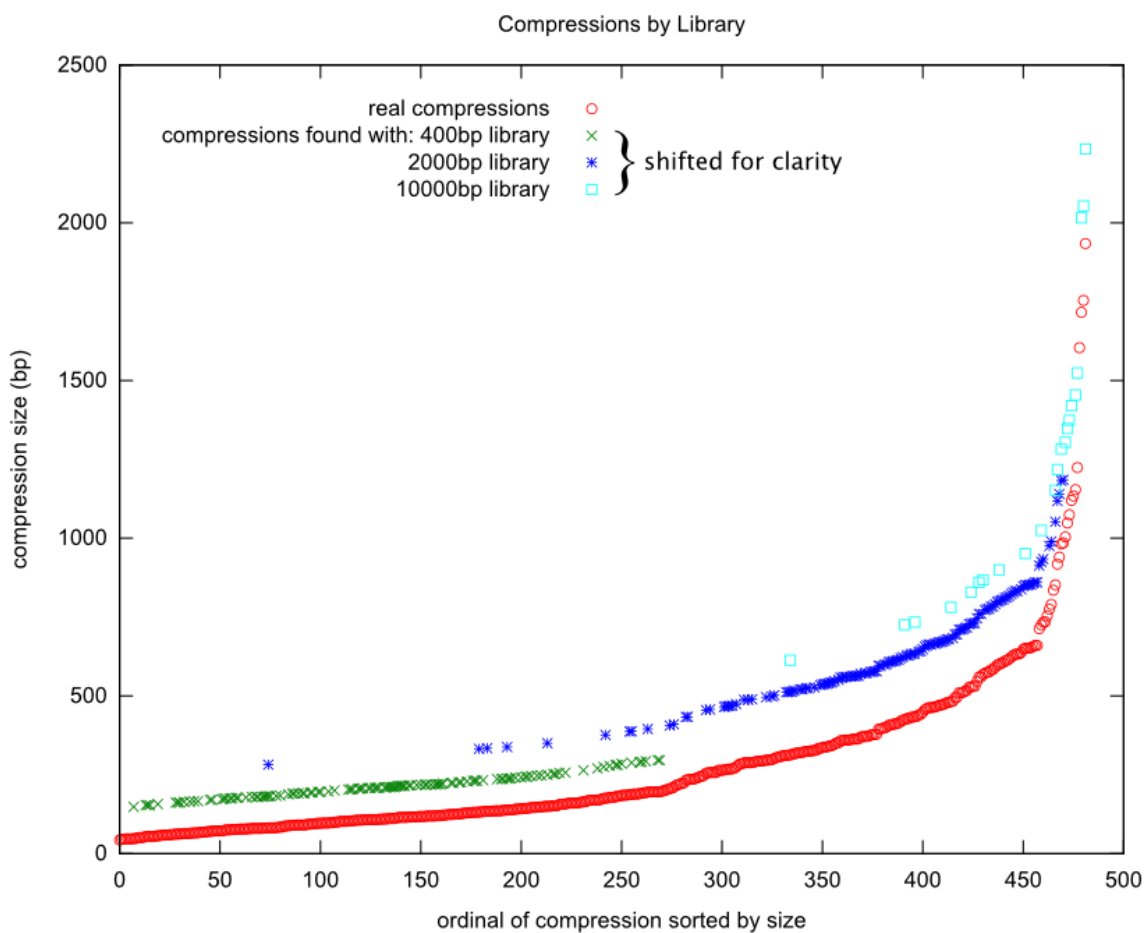


Figure 23: **Compressions found using three libraries.** We created faux reads from human chromosome 1 sequence. We created three libraries of mated reads and one library of unmated reads. Using the Celera

Assembler 5.4, we created a (draft) assembly, which turned out to have 482 compressions (red circles). We were able to identify 284 of them using one or more of the libraries. The compressions range from 40 bp to 1935 bp (vertical axis). We numbered the compressions, sorted by size, from smallest to largest (horizontal axis). When a red circle indicates a compression that was identified by one of the three libraries, this is indicated by symbols shifted upward (for the sake of clarity) from the red circle. We required that the position and size of the compression be correctly estimated (see text).

**Working with short read data.** To test the Shooting Algorithm, we used the largest scaffold in the assembly. This scaffold was 28.1 Mbp long and contained 584 contigs. One of the problems we faced in working on an assembly made from short read data, is that these assemblies usually have very high read coverage. This can make our local pools quite large, and computing overlaps between the reads in the pools can become infeasible especially for reads from highly repetitive portions of the genome. Therefore, we chose to only close gaps less than 2000 bp in size. The other modification we made was to allow a minimum of 25 bp overlap between reads when computing the paths. (This is lower than the minimum 40 bp we used for the bacterial assemblies and *Bos taurus* which were sequenced with longer Sanger reads).

**Results.** There were 517 gaps in the scaffold that were less than 2000 bp. Of these, we were only able to find a path for 34 of these. The major reason for the inability of the Shooting Algorithm to find a path was the fact that we only used mates in the contigs flanking the gap for building local pools of reads. In 407 of the gaps, one of the flanking contigs was much shorter than the 2000 bp, and therefore no mates of inserts from the 2 Kbp and 10 Kbp libraries could be used. This shortcoming in our method could be overcome by using inserts whose mates may be in nearby contigs, not just those in the contigs directly adjacent to the gap. Of 517 gaps, 110 were flanked by two contigs both of which were longer than 2Kbp.

Our CE statistic in combination with the Shooting Algorithm identified 48 suspected CE misassemblies. Of these, 24 (50%) were real misassemblies. We were able to find the correct sequence for 7 of the 24. The total number of real misassemblies was 47, and therefore we missed 23 of them.

## APPENDIX A: THE ALLPATHS ALGORITHM

In this section we discuss the similarities and differences between our Shooting Method and the ALLPATHS (Maccallum et al. 2009), assembler. Like the Celera Assembler, it is a *de novo* assembler, creating assemblies from read and mate pair data without using a reference genome. As of this writing, there have been no published papers reporting genomes created with ALLPATHS. The ALLPATHS assembler includes a module which seeks to identify all possible paths of reads spanning a mate pair. The ALLPATHS assembler also uses mate pairs to build local clouds of reads similar to the pools used in the Shooting Method.

While this step is similar to our technique, there are some differences. This is a complicated program which we cannot hope to do justice here. Hence, we discuss some features, and the reader should recognize that we take shortcuts and do not tell the whole story. ALLPATHS chooses an integer  $k > 0$ , determines all *k-mers*, strings of length  $k$ , in the collection of reads, and creates a directed graph (de Bruijn graph) in which the  $k$ -mers are nodes, and there is an edge between two  $k$ -mers if there is a read in which the two  $k$ -mers occur, one being shifted from the other by one base (hence, the last  $k-1$  bases of one will be the first  $k-1$  bases of the other). Hence, ALLPATHS does not compute overlaps between reads.

A big difference between the two algorithms is our goals. The goal of ALLPATHS technique is to create a *de novo* assembly, whereas our technique is a precise post-processor aimed at extracting all possible information from existing read data to close gaps and correct CE misassemblies in the assembly and therefore save time and effort used for finishing the genome.



Another difference is that ALLPATHS never explicitly computes all overlaps between the reads in the cloud. The ALLPATHS algorithm constructs the paths using minimal extensions of the reads found using a technique for k-mer database lookups done on the fly. The reason ALLPATHS does not compute overlaps is that their clouds of reads are big, encompassing relatively large (on the order megabases) regions of the genome. We construct a much more localized pool of reads for each gap, and thus our pools are much smaller, allowing us to compute overlaps with a smaller likelihood that an overlap will be spurious.

Finally, an important difference between the two algorithms is that ALLPATHS is only applicable to short read assembly with reads of length 25-50 bp. It is not applicable to longer reads over a 100 bp such as Sanger sequenced reads or reads made with the 454 technology. Ideally, the k-mer size,  $k$ , should be a bit less than the read size since if  $k$  is much less than the read size, a great deal of information is left out of the de Bruijn graph. Paths are allowed by the de Bruijn graph which are not allowed in the genome whenever there is a repeat of size at least  $k$  which is spanned by a read. While, technically, one can use the reads to weed out these paths, this work becomes a large factor in the assembly process. If the reads are long (perhaps 100 bp), and  $k$  is perhaps 90, then a large fraction of the 90-mers may contain sequencing errors. While ALLPATHS tries to do error correction of reads or k-mers, the process becomes much more difficult if most of the k-mers occur only once in the set of reads. To circumvent this, it is possible to generate much deeper read coverage, but then one needs more memory to do error correction, and the reason for using the de Bruijn approach is to reduce memory needs. Better algorithms may permit effective use of large  $k$ . In particular, it is unlikely that this approach could

be used effectively when some of the read data is Sanger read data. The Shooting Method is applicable to reads of any size and produced by any technology.

## Bibliography

Anon.n.d. Genome.gov | Human Genome Sequence Quality Standards.

<http://www.genome.gov/10000923>.

Aparicio, Samuel, Jarrod Chapman, Elia Stupka, Nik Putnam, Jer-Ming Chia, Paramvir Dehal, Alan Christoffels, et al. 2002. Whole-genome shotgun assembly and analysis of the genome of *Fugu rubripes*. *Science (New York, N.Y.)* 297, no. 5585 (August 23): 1301-1310. doi:10.1126/science.1072104.

Batzoglou, Serafim, David B Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, Evan Mauceli, Bonnie Berger, Jill P Mesirov, and Eric S Lander. 2002. ARACHNE: a whole-genome shotgun assembler. *Genome Research* 12, no. 1 (January): 177-189. doi:10.1101/gr.208902.

Celniker, Susan, David Wheeler, Brent Kronmiller, Joseph Carlson, Aaron Halpern, Sandeep Patel, Mark Adams, et al. 2002. Finishing a whole-genome shotgun: Release 3 of the *Drosophila melanogaster* euchromatic genome sequence. *Genome Biology* 3, no. 12: research0079.1-0079.14. doi:10.1186/gb-2002-3-12-research0079.

Deshayes, Caroline, Emmanuel Perrodou, Sebastien Gallien, Daniel Euphrasie, Christine Schaeffer, Alain Van-Dorsselaer, Olivier Poch, Odile Lecompte, and Jean-Marc Reytrat. 2007. Interrupted coding sequences in *Mycobacterium smegmatis*: authentic mutations or sequencing errors? *Genome Biology* 8, no. 2: R20. doi:10.1186/gb-2007-8-2-r20.

Genome Reference Consortium. 2010. Genome Reference Consortium: human. *Genome Reference Consortium: human*. October 20.

<http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/human/index.shtml>.

- Havlak, Paul, Rui Chen, K. James Durbin, Amy Egan, Yanru Ren, Xing-Zhi Song, George M. Weinstock, and Richard A. Gibbs. 2004. The Atlas Genome Assembly System. *Genome Research* 14, no. 4 (April): 721-732. doi:10.1101/gr.2264004.
- Huang, Xiaoqiu, Jianmin Wang, Srinivas Aluru, Shiaw-Pyng Yang, and LaDeana Hillier. 2003. PCAP: A Whole-Genome Assembly Program. *Genome Research* 13, no. 9: 2164-2170. doi:10.1101/gr.1390403.
- Kurtz, S., A. Phillippy, A. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. Salzberg. 2004. Versatile and open software for comparing large genomes. *Genome biology* 5, no. 2: R12.
- Lander, Eric S., and Michael S. Waterman. 1988. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics* 2, no. 3 (April): 231-239. doi:10.1016/0888-7543(88)90007-9.
- Li, Ruiqiang, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, et al. 2010. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research* 20, no. 2 (February 1): 265 - 272. doi:10.1101/gr.097261.109.
- Maccallum, Iain, Dariusz Przybylski, Sante Gnerre, Joshua Burton, Ilya Shlyakhter, Andreas Gnirke, Joel Malek, et al. 2009. ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology* 10, no. 10: R103. doi:10.1186/gb-2009-10-10-r103.
- Mandel, Mark, Eric Stabb, and Edward Ruby. 2008. Comparative genomics-based investigation of resequencing targets in *Vibrio fischeri*: Focus on point miscalls

and artefactual expansions. *BMC Genomics* 9, no. 1: 138. doi:10.1186/1471-2164-9-138.

Margulies, Marcel, Michael Egholm, William E. Altman, Said Attiya, Joel S. Bader, Lisa A. Bemben, Jan Berka, et al. 2005. Genome sequencing in microfabricated high-density picolitre reactors. *Nature* 437, no. 7057: 376-380. doi:10.1038/nature03959.

Myers, Eugene W., Granger G. Sutton, Art L. Delcher, Ian M. Dew, Dan P. Fasulo, Michael J. Flanigan, Saul A. Kravitz, et al. 2000. A Whole-Genome Assembly of *Drosophila*. *Science* 287, no. 5461 (March 24): 2196-2204. doi:10.1126/science.287.5461.2196.

National Human Genome Research Institute. 2009. Genome.gov | Human Genome Sequence Quality Standards. *Genome.gov | Human Genome Sequence Quality Standards*. July 31. <http://www.genome.gov/10000923>.

Roberts, Michael, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. 2004. Reducing storage requirements for biological sequence comparison. *Bioinformatics* 20, no. 18 (December 12): 3363-3369. doi:10.1093/bioinformatics/bth408.

Salzberg, Steven L., and James A. Yorke. 2005. Beware of mis-assembled genomes. *Bioinformatics* 21, no. 24 (December 15): 4320-4321. doi:10.1093/bioinformatics/bti769.

Schatz, Michael C., Arthur L. Delcher, and Steven L. Salzberg. 2010. Assembly of large genomes using second-generation sequencing. *Genome Research* 20, no. 9: 1165 - 1173. doi:10.1101/gr.101360.109.

- Shendure, Jay, Gregory J. Porreca, Nikos B. Reppas, Xiaoxia Lin, John P. McCutcheon, Abraham M. Rosenbaum, Michael D. Wang, Kun Zhang, Robi D. Mitra, and George M. Church. 2005. Accurate Multiplex Polony Sequencing of an Evolved Bacterial Genome. *Science* 309, no. 5741 (September 9): 1728-1732. doi:10.1126/science.1117389.
- Sutton, Granger G., Owen White, Mark D. Adams, and Anthony R. Kerlavage. 1995. TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Science and Technology* 1, no. 1 (January 1): 9-19. doi:10.1089/gst.1995.1.9.
- The Bovine Genome Sequencing and Analysis Consortium, Christine G. Elsik, Ross L. Tellam, and Kim C. Worley. 2009. The Genome Sequence of Taurine Cattle: A Window to Ruminant Biology and Evolution. *Science* 324, no. 5926 (April 24): 522-528. doi:10.1126/science.1169588.
- The International HapMap Consortium. 2007. A second generation human haplotype map of over 3.1 million SNPs. *Nature* 449, no. 7164 (October 18): 851-861. doi:10.1038/nature06258.
- Wild, C. J. 2000. *Chance Encounters: A First Course in Data Analysis and Inference*. New York: John Wiley.
- Zimin, Aleksey, Arthur Delcher, Liliana Florea, David Kelley, Michael Schatz, Daniela Puiu, Finnian Hanrahan, et al. 2009. A whole-genome assembly of the domestic cow, *Bos taurus*. *Genome Biology* 10, no. 4: R42. doi:10.1186/gb-2009-10-4-r42.
- Zimin, Aleksey V., Douglas R. Smith, Granger Sutton, and James A. Yorke. 2008. Assembly reconciliation. *Bioinformatics* 24, no. 1 (January 1): 42-45.

[doi:10.1093/bioinformatics/btm542](https://doi.org/10.1093/bioinformatics/btm542).