INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

ALGORITHMS FOR WHOLE GENOME SHOTGUN SEQUENCING

by

Eric Anson

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

**In Partial Fulfillment of the Requirements
For the Degree of**

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

1999

UMI Number: 9965855

UMI[®]

UMI Microform 9965855

Copyright 2000 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

THE UNIVERSITY OF ARIZONA ©
GRADUATE COLLEGE

As members of the Final Examination Committee, we certify that we have
read the dissertation prepared by Eric Anson
entitled Algorithms for Whole Genome Shotgun Sequencing

and recommend that it be accepted as fulfilling the dissertation
requirement for the Degree of Doctor of Philosophy

Eugene Myers

11/12/99

Date

Peter Downey

11/12/99

Date

William Evans

11/12/99

Date

Bruce Bayly

Date

12 November 1999

Date

Final approval and acceptance of this dissertation is contingent upon
the candidate's submission of the final copy of the dissertation to the
Graduate College.

I hereby certify that I have read this dissertation prepared under my
direction and recommend that it be accepted as fulfilling the dissertation
requirement.

Dissertation Director Eugene Myers

11/12/99

Date

STATEMENT BY AUTHOR

This Dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED:

TABLE OF CONTENTS

1	LIST OF FIGURES	6
2	LIST OF TABLES	7
3	ABSTRACT	8
4	INTRODUCTION.....	10
	4.1. DNA	10
	4.2. SEQUENCING SMALL STRANDS OF DNA	12
	4.3. BASIC SHOTGUN SEQUENCING	17
	4.4. DOUBLE-BARRELED SHOTGUN SEQUENCING	26
	4.5. THE HUMAN GENOME PROJECT	29
	4.6. WHOLE GENOME SHOTGUN SEQUENCING AND THE SEQUENCING RACE	33
	4.7. WHAT'S IN THIS THESIS.....	40
5	DATA STRUCTURES AND BASIC MOVES.....	43
	5.1. THE PROBLEM	43
	5.2. DATABASE QUERY	46
	5.3. REPEAT EDGE DETECTION.....	47
6	THE SIMULATOR	52
	6.1. WHAT IS SIMULATED.....	52
	6.2. SIMULATOR PRIMITIVES AND REPEAT EDGE DETECTION	54
7	THE QUICK SCAN ALGORITHM.....	58
	7.1. BASIC IDEAS.....	58
	7.2. THE FOOTPRINT.....	60
	7.3. THE TAKE STEP PROCEDURE	61
	7.4. EXPAND A FOOTPRINT	64
	7.5. THE QUICK SCAN SEARCH	70
8	THE INTER-MARKER ASSEMBLY ALGORITHM	76
	8.1. REPETITIVE FOOTPRINTS	76
	8.2. THE FIRST PASS.....	80
	8.3. THE BACKTRACKING SEARCH	83
9	EMPIRICAL RESULTS	90

TABLE OF CONTENTS - Continued

10	THE CELERA GENOME SEQUENCING PROJECT	100
	10.1. THE SHOTGUN STEP	100
	10.2. THE CELERA ASSEMBLER AND UNITIGS	104
	10.3. CELSIM: CELERA'S SIMULATOR	109
	10.4. APPLYING THE ALGORITHMS.....	112
11	CONCLUSION	125
12	REFERENCES	130

1 LIST OF FIGURES

FIGURE 4-1: A SIMPLE DNA MODEL	11
FIGURE 4-2: AUTORADIOGRAM OF SEQUENCING GEL.....	14
FIGURE 4-3: CREATING PREFIX ENDING IN G	16
FIGURE 4-4: HOW A REPEAT CAN CAUSE A FALSE OVERLAP.....	22
FIGURE 4-5: AN EXAMPLE OF THE SHORTEST ASSEMBLY NOT BEING THE BEST	24
FIGURE 4-6: THE HGP TWO-TIERED APPROACH.....	30
FIGURE 5-1: THE THREE TYPES OF INSERTS USED IN OUR MODEL	45
FIGURE 5-2: REPEAT EDGE DETECTION.....	47
FIGURE 5-3: THE DYNAMIC PROGRAMMING MATRIX TO FIND REPEAT EDGES.....	49
FIGURE 7-1: WALK ACROSS THE MARKERS.....	59
FIGURE 7-2: HOW REPEATS CAN THROW A WALK OFF COURSE.....	60
FIGURE 7-3: DETECTING THAT R' IS IN A REPEAT BY INCONSISTENCY WITH S AND T	63
FIGURE 7-4: EXPANDING OVER A REPEAT.	67
FIGURE 8-1: DIAGRAM OF REPETITIVE FOOTPRINT.....	78
FIGURE 8-2: TWO REPEATS KEEP WALKS FROM MEETING AFTER FIRST PASS.....	83
FIGURE 8-3: A WALK THAT REQUIRES MULTIPLE LEVELS OF BRANCH SEARCHES.....	84
FIGURE 8-4: A WALK JOINED AFTER THE FIRST PASS.	88
FIGURE 10-1: NORMAL UNITIGS FAIL TO SPAN A SHORT REPEAT.....	107
FIGURE 10-2: HOW A MAXIMAL CONNECTED SUBGRAPH MAY NOT COVER ALL UNIQUE PORTIONS OF THE SOURCE.	118
FIGURE 10-3: A REPETITIVE FOOTPRINT MODELING A FRACTURED REPEAT	119
FIGURE 10-4: A REPEAT BLIZZARD	120

2 LIST OF TABLES

TABLE 1: ALIGNMENT SCORE PER LENGTH FOR RANDOM SEQUENCES	51
TABLE 2: AVERAGE DATABASE QUERIES FOR THE DIFFERENT MEETING STRATEGIES.....	73
TABLE 3: QUICK SCAN SUCCESS RATE WITH VARYING COVERAGE	92
TABLE 4: QUICK SCAN SUCCESS RATE FOR DIFFERENT DISTANCES BETWEEN MARKERS.....	92
TABLE 5: QUICK SCAN SUCCESS RATE WITH VARYING RATIO OF LONG TO SHORT FRAGMENTS	93
TABLE 6: QUICK SCAN SUCCESS RATE WITH VARYING CLUSTERING AND A CLUSTER DENSITY OF 1.5.....	93
TABLE 7: QUICK SCAN SUCCESS RATE WITH VARYING CLUSTERING AND A CLUSTER DENSITY OF 2.0.....	94
TABLE 8: QUICK SCAN SUCCESS RATE OVER INTERVALS THAT CONTAINED A LONG REPEAT	95
TABLE 9: QUICK SCAN SUCCESS RATE USING MORE COMPLEX REPEAT STRUCTURE	96
TABLE 10: SUCCESS RATE OF INTER-MARKER ASSEMBLY ALGORITHM	96
TABLE 11: PERCENT OF UNIQUE SEQUENCE COVERED BY INTER-MARKER ASSEMBLY ALGORITHM.....	98
TABLE 12: PERCENT OF SEQUENCE CHARACTERIZED	98
TABLE 13: PERCENT OF SOURCE UNIQUELY SEQUENCED	99
TABLE 14: SUCCESS RATE USING OF INTER-MARKER ASSEMBLY USING CELERA'S INSERT MODEL.....	102
TABLE 15: PERCENT OF UNIQUE SEQUENCE COVERED BY INTER-MARKER ASSEMBLY ALGORITHM USING THE CELERA MODEL.....	103
TABLE 16: PERCENT OF SEQUENCE CHARACTERIZED USING CELERA MODEL	103
TABLE 17: PERCENT OF SOURCE UNIQUELY SEQUENCED USING CELERA MODEL	104

3 ABSTRACT

A monumental achievement in the history of science, the sequencing of the entire human genome, will soon be reached. The Human Genome Project (HGP) has been working toward this goal since 1990 using a two-tiered strategy. Recently it was proposed that using a whole-genome shotgun approach to sequence the genome would be faster and less costly. This thesis expands on that proposal by presenting two algorithms that can be used in whole-genome shotgun sequencing. These algorithms were implemented and tested on simulated data.

Essential to this approach is the availability of pairs of short, unique sequence markers at a roughly estimated distance from each other. Determining the sequence of the genome can then be broken into a series of inter-marker assembly problems that determine the sequence between a pair of markers. Unfortunately, marker pairs are not always correct and repeats can greatly confound the assembly. This motivates the first problem of rapidly finding a set of linked contigs, called a scaffold, between a pair of markers that confirms the marker pair and the ability to traverse the region between them. Then an inter-marker assembly algorithm that determines the unique sequence segments between a marker pair is presented. Both algorithms are evaluated with respect to a simulation that can model various types of repeats and for which our only information about the presence of repeats is excessive coverage and the ability to detect their boundaries. Simulation results show that at 10x coverage one can find and assemble the unique sequence between markers more than 99.9% of the time for many of the repeat models.

Events in this field have been moving rapidly. Recently a new company called Celera Genomics announced its intention to sequence the human genome before the HGP by using the whole-genome shotgun approach. We end this thesis by briefly discussing Celera's approach, and relating it to the algorithms presented here.

4 INTRODUCTION

4.1 DNA

Few people in the United States have not heard of DNA. This acronym seems to appear everywhere. It crops up in works of popular fiction, comic books, TV shows, and movies. It played a key role in the murder trial of O.J. Simpson. Stories like the cloning of the sheep Dolly permeate the news media. All this media attention is not unwarranted. We are in the midst of a molecular biology revolution. At the end of 1997, the world's farms were growing over thirty million acres of crops that were genetically altered to combat insects, disease, or to increase their yield¹. Medicines, such as insulin, are manufactured using genetic engineering. The not too distant future could see such heady developments as the understanding and perhaps control of all diseases. Other conditions such as obesity, baldness, and even aging might be arrested. The evolutionary history of life on earth could be unraveled. In the next few years a key step to these accomplishments and a monumental event in man's history will be achieved; the sequencing of the entire human genome.

Every living creature contains a set of genetic instructions telling how it should be built and operated. This set of instructions, known as the *genome*, is replicated in every cell of the organism, though different parts of the instructions, known as *genes*, are expressed in different types of cells. In humans, these instructions are carried on 24 chromosomes (23 for females). Each living cell in the human body (with the exception of the red blood cells) has two copies of each of the 22 nonsex chromosomes, an X

chromosome, and a Y chromosome for males or a second X chromosome for females. These chromosomes, which instruct each cell in our body how to function, are molecules of DNA.

DNA, or deoxyribonucleic acid, is a polymer made up of a linear arrangement of subunits known as nucleotides. Each nucleotide has three components: a phosphate group, a sugar group, and a nitrogen-containing base. The sugar is always deoxyribose, so the nucleotide types differ only in the nature of their base. In DNA, there are four alternative bases: adenine, thymine, guanine, and cytosine, which we will refer to as A, T, G, and C respectively.

Deoxyribose contains five carbon atoms that are numbered 1' to 5'. In DNA a phosphate group connects the 3' carbon of one sugar to the 5' carbon of its neighbor, thus

forming a directed sequence, or strand, of nucleotides. Normally a DNA molecule consists of two of these strands joined together and wound around in a helical arrangement. This is the famous DNA *double helix* first proposed by Watson and Crick².

In double stranded DNA, the two strands have opposite orientation and the bases of one strand are paired (via hydrogen bonds) with the bases of the other strand. Adenine (A) in one strand is always paired with thymine (T) in the other and guanine (G) is always paired with cytosine (C). This kind of pairing is termed *complementary base pairing* and we call the paired bases the *complements* of each other. (E.g. T is the complement of A. See Figure 4-1.) Notice that if the sequence of one strand is known, then the sequence of the other can be deduced. Thus a DNA molecule can be completely specified by a sequence over a four-character alphabet. Determining this sequence for the 3 to 4 billion base pairs of the human genome is the goal of whole genome shotgun sequencing.

4.2 Sequencing Small Strands of DNA

Methods for determining the sequence of bases which comprise a given strand of DNA were devised in the late 70's by Allan Maxam and Walter Gilbert in the United States³, and Fredrick Sanger and his coworkers in England⁴. These techniques, for which the inventors were jointly awarded the Nobel Prize, still form the basis of all current sequencing methods. Though these methods do vary somewhat, almost all follow the basic scheme outlined below.

First, a set of all prefixes of the target DNA is created. These smaller pieces of DNA are produced so that the last base in the sequence is known. They are also marked with a radioisotope or fluorescent dye so they can be tracked. Next these pieces are size-sorted through a process known as *gel electrophoresis*. They are placed in a permeable gel across which a voltage is applied. DNA molecules, being negatively charged due to their phosphate groups, migrate toward the positive charge. Since the charge per unit length of the pieces of DNA is constant, one might expect the different pieces to travel through the gel at the same rate. The fibrous nature of the gel, however, hinders the progress of the larger molecules, so shorter pieces of DNA travel faster than longer pieces. Thus, since the end base of each piece is known, the sequence of the target DNA can be determined by noting the distance traveled by these pieces.

An example of this process is illustrated in Figure 3-2. In this example the gel is prepared with four parallel lanes. The radio-marked pieces of DNA are in separate solutions depending on their end base. Each solution is placed at the top of the corresponding lane in the gel. A voltage is then applied across the gel and the pieces of DNA migrate toward the positive charge. After a set period of time the voltage is removed, and the radio-labeled fragments are visualized by exposing the gel to an x-ray filter to make an *autoradiogram* (illustrated on the right side of Figure 3-2). Each band in the figure represents all prefixes that are the same length. Since shorter pieces travel through the gel faster than longer pieces, the original sequence can be read from the bands moving from the bottom of the figure to the top.

In gel electrophoresis the mobility of each piece of DNA within the gel varies as the reciprocal of the logarithm of its length. That is, the separation of pieces of consecutive lengths approaches zero as their sizes increase. This effectively limits the size of DNA strands that can be sequenced via this method. With today's technology an average of 500 bases of the source fragment can be determined, with an upper limit of 800-900 bases for an unusually clean reaction. A strand of DNA so sequenced is called a *read*.

Machines to automate this process have been on the market for a number of years. Many of these machines require that the fragments be labeled with a fluorescent dye rather than radioisotopes. A different color dye is used for each of the prefixes ending in

each of the four bases. This alleviates the need to use separate lanes to differentiate the bases. The machines employ laser-induced fluorescence to automatically detect the order of the labeled fragments as they migrate through the gel. In *slab gel* machines, the gel is sandwiched between two glass plates. Many reads are sequenced down separate lanes of the gel. The number of lanes used by these machines varies greatly, but 16 to 64 lanes per run are typical numbers. The latest machines are called *capillary gel* machines. In these machines, the gel slab is replaced by gel-filled glass tubes, each about the width of a human hair. Because of their small volume, the capillaries are better at dissipating heat, which allows them to be run at higher electric fields than slab gels, making them achieve their results faster. Capillary gel machines also eliminate lane tracking problems (this will be discussed later) and allow more reads to be sequenced per run. For example the Perkin-Elmer ABI Prism 3700 has 96 capillaries.

At this point one may wonder how the prefixes used in sequencing are created. The most common method used is called the *chain termination method* or the *dideoxy* method. The process is illustrated in Figure 3-3. Start with a single stranded copy of the DNA molecule to be sequenced. We will call this molecule the *template*. A *primer* binds to the 3' end of the template. A primer is a short stretch of nucleic acid needed to get synthesis of a new strand of DNA started. The primer allows an enzyme called *DNA polymerase* to bind to the template. The polymerase travels along the template creating a second strand of DNA by facilitating the binding of complementary bases. This is part of the process used by cells to duplicate their DNA before cellular division.

Whenever a base is added to a growing strand of nucleic acid it is provided as the *nucleoside triphosphate* (NTP), a base linked to a sugar and three phosphate groups. In normal DNA synthesis the sugar is deoxyribose and the molecule required is *deoxynucleoside triphosphate* (dNTP). If instead the sugar is dideoxyribose, which looks like deoxyribose except that it lacks the hydroxyl group off the 3' carbon, then the

molecule is called *dideoxynucleoside triphosphate* (ddNTP). DNA polymerase will use ddNTP to add a base to the strand of DNA being created, however, the missing hydroxyl group will prevent further bases from being added and the synthesis will terminate at that point.

Thus the desired prefixes are created in four solutions. Each solution contains multiple copies of the template, the primer, and dNTP molecules. We also include a different type of ddNTP molecules in each solution. That is, each solution contains ddATP, ddCTP, ddGTP, or ddTTP molecules. After the reactions complete these solutions will contain the prefixes ending in A, C, G, and T respectively. Figure 4-3 illustrates a solution containing ddGTP. When ddGTP is used to add a base to the growing sequence the process terminates. This is represented by the underlined character in the sequence.

4.3 Basic Shotgun Sequencing

To determine the sequence of longer stretches of DNA, Sanger and his cohorts⁵ devised what is called the *shotgun DNA sequencing* strategy. This method is at the heart of all current approaches to genome sequencing. At its most basic level, shotgun sequencing involves randomly breaking many copies of the target DNA into fragments small enough to be sequenced as described in Section 4.2. Overlaps between these sequences are then searched for and used to put the pieces of this ‘puzzle’ back together. We will now examine this procedure in more detail.

Given many copies of a target molecule of DNA, the first step is to fracture them into smaller pieces in a uniformly random manner. This is usually accomplished by using sound, i.e. sonication, or by being passed through a nozzle under pressure, i.e. nebulation. Of the fragments thus obtained, those that are a desired length are chosen. This is done through gel electrophoresis, using only those fragments that lie within a particular band of the gel after a set period of time. With this process it is possible to select fragments whose sizes form a normal distribution around the desired length with about a 10% variance. The chosen fragments are then inserted into a *vector*, which is an engineered molecule of DNA that can replicate and copy, or *clone*, the fragment. For this reason these fragments are often called *inserts*. Usually the vector is the DNA of a genetically engineered bacterial virus (*bacteriophage* or *phage*). These viruses are then used to infect bacteria colonies (one virus per colony). This produces thousands of copies of the virus, and thus the associated insert. These viruses are then extracted. The viruses have been genetically engineered to have a primer at the position where the target insert is spliced in. This allows the insert to be sequenced by the method described in Section 3.3. The size of the insert is chosen to be larger than the maximum possible read, so that just the sequence on the right or left end of the insert is read.

The problem now passes out of the hands of the biologists and enters into the domain of the mathematicians and computer scientists. Suppose one started with a source of DNA that had length $G = 100\text{K}$ bases. The procedure above generates R reads with an average length L , making a total of $N = R \times L$ bases read. The value of $c = \frac{N}{G}$ is called the average sequencing *coverage*. Typically one determines a desired coverage

before hand, and then sequences enough reads to obtain that coverage. For example, suppose $L = 500$, and a coverage of 10 (a very typical value) is desired, then

$$R = \frac{cG}{L} = 2000 \text{ reads must be sequenced.}$$

Determining the sequence of the source DNA from these reads is known as the *fragment assembly problem*.

Many software packages have been developed to assemble such a set of reads. Most divide the problem into three phases: *overlap*, *layout*, and *consensus*.^{6, 7, 8} The overlap phase builds an *overlap graph* by comparing every read with every other read to find all overlaps of some minimal length. We choose the minimal length of overlaps to be large enough to avoid reporting overlaps that occur by chance. Throughout this thesis we will use the label *MIN_OVERLAP* for the length of the smallest overlap to be considered. We will also sometimes use the shorthand Δ for *MIN_OVERLAP*. A typical value for *MIN_OVERLAP* is 35-50 bases. The vertices of the overlap graph are the reads and the edges represent overlaps between them.

The software for building the overlap graph must account for a couple of characteristics of the data that are due to its biological origin. First, since the source DNA molecule is dual stranded, one doesn't know which strand the read came from. The solution is to compare both the fragment and its Watson-Crick complement against all the other fragments. The Watson-Crick complement of a sequence is the sequence in reverse order with all of its bases replaced by their complements. In other words, Watson-Crick complement is the sequence that would be opposed to the fragment in a dual-stranded DNA molecule. This doubles the number of comparisons that are required to form the

overlap graph. It also means that the overlap graph must keep track of the relative orientation of the reads that overlap.

A factor that does complicate the process of building the overlap graph is *sequencing error*. As is true with most physical experiments, the method of sequencing a read via gel electrophoresis is prone to error. These errors can come in the form of added bases (insertions), missed bases (deletions), or misread bases (substitutions). As would be expected, the error rate increases toward the end of the read as the separation in the gel shrinks. The error rate in the good portion of the read (the first 300-500 bases) is usually about 1%-5% depending on the lab doing the sequencing. After that the error rate increases rapidly, quickly making the rest of the read unusable. These sequencing errors mean that one cannot search for exact overlaps but must use more computationally expensive algorithms for finding approximate matches.

Due to sequencing errors, the overlap phase must search for ϵ -overlaps. These are overlaps where the rate of differences is bounded by ϵ . That is, if the overlap reported between two reads has length X , then the differences between the two sequences covered by the overlap must not exceed $\epsilon \times X$. This problem has been much studied and deterministic algorithms exist⁹ which solve it efficiently for good size problems. For especially large problems, however, researchers often resort to heuristic methods. Typically this involves searching the reads for exact matches of some length n using a hash table. This process is used to screen out those reads that are to be investigated for potential overlaps. The value of n is chosen to balance the need between speed and

sensitivity. The value of ϵ greatly influences the choice of n . Larger sequencing errors dictate smaller values for n or an increase in the number of ϵ -overlaps missed.

In the layout phase, relative locations for the reads are chosen. Ideally, these would match their true locations in the source sequence of DNA. This placement of the reads is called a *layout*. The layout is completely determined by a spanning forest of the overlap graph as such a subset positions every read with respect to every other, transitively, through the overlaps on the path between them. We will add the requirement that the overlaps transitively implied by such a spanning tree do in fact exist. For example, suppose that in a particular layout reads R_1 and R_2 have overlaps on the right side of read R_3 , then an overlap between R_1 and R_2 should exist. Even with this restriction, there are many different layouts possible. The source of this ambiguity is *repeats* in the source DNA sequence.

A repeat in the source DNA is a subsequence that appears, with some degree of variation, multiple times throughout the sequence. Three measurements characterize repeats: *length*, *frequency*, and *fidelity*. Length measures the size of the repeat in bases. This can vary from the 300-500 base length of a common type of repeat, called an Alu, in the human genome, to large chromosome duplications that can be 50-150K bases long. Frequency measures how often the repeat occurs in the target sequence. It is estimated that the human genome contains one million Alu's. We will call repeats like Alu's that compose at least 1% of the genome *ubiquitous* repeats. On the other hand an estimated 80,000 genes are thought to be repeated 1-4 times each. Repeats like these that have fewer than 10 occurrences will be called *low copy number* repeats. Lastly, fidelity

measures how much each occurrence of the repeat differs from each other. If the fidelity is low enough, the repeat ceases to be a problem. However, repeats exist that are almost exact copies of each other.

Repeats may be scattered roughly uniformly across the genome, or they may be clustered together. A repeat with multiple occurrences lying next to each other is called a *tandem repeat*. Tandem repeats may involve a few copies (2-5) of a medium or large sequence. Nasty tandem repeats exist, however, that involve a small sequence (e.g. 1-3 bases) repeated hundreds or even thousands of times. This type of tandem repeat is extremely difficult to sequence and often the portions of the genome containing them are ignored. This is not largely detrimental since tandem repeats involving very small sequences are not biologically interesting.

If the source DNA sequence contained no repeats, then there would be only one possible layout, the layout that agrees with the source. Repeats cause reads that do not come from overlapping portions of the source sequence to appear to overlap. This is illustrated in Figure 4-4. Although reads A and B come from distant parts of the source sequence, they appear to overlap because their right and left ends respectively contain

matching sequence. These false overlaps lead to multiple possible layouts, and dictates the need to establish some criteria to choose one. Traditionally the layout that results in the shortest sequence has been sought. Unfortunately, this criterion results in layouts that do not match the source. (See Figure 3-5) Additionally, finding a spanning tree that maximizes some criterion such as shortest or most-likely is known to be NP-hard.¹⁰ This has lead investigators to propose various approximate algorithms including greedy algorithms that come within a given factor of optimal^{11, 12}, simulated annealing algorithms, relaxation methods based on weighted matchings in order of score¹³, and a reduction to a greedy Eulerian tour¹⁴.

Finally, the consensus phase attempts to reconstruct the source sequence from the layout. Redundancy in the sequencing is used to pinpoint the sequencing errors in the individual reads. This requires an optimal multiple alignment of all the reads. Computing an optimal multiple alignment requires time that is exponential in the depth of the coverage. Thus heuristic methods are used in practice. Most of these methods start with an alignment induced by a pair-wise merging of the overlaps in the spanning forest obtained by the overlap phase¹⁵. This alignment is then refined using a window-sweep optimization, a Hidden-Markov model gradient descent algorithm¹⁶, or a round-robin realignment¹⁷.

There is one last difficulty to determining the target DNA through shotgun sequencing. Even though many more bases have been sequenced than the length of the source DNA, the stochastic nature of the sampling dictates that all bases of the source may not be covered. This means that even in the absence of sequencing errors and repeats, the solution of shotgun sequencing will be fragmented and incomplete. The result of the layout phase is a series of contiguous overlapping reads called *contigs*. Each tree in the spanning forest produced by the layout phase defines a contig. A multiple alignment for each of these contigs is then produced by the consensus phase to determine

a set of disjoint subsequences of the source. The portions of the source sequence not covered by reads are called *gaps*. A stochastic analysis of this process by Lander and Waterman¹⁸ determined the following expected values:

- Fraction of source covered by some read: $1 - e^{-c}$
- Number of gaps: Re^{-c}
- Average contig length: $(L/c)e^c$
- Average length of gap: L/c

Recall that c is the coverage, L is the average length of a read, and R is the number of reads. These values were calculated assuming that the sampling of the source was perfectly uniform, which is not an unreasonable assumption.

The source DNA length, G , has increased rapidly over the past few years. When shotgun sequencing was first introduced in the early 1980's a typical value for G was 3-5K bases. By 1990, cosmid-sized (approximately 40K bases) fragments of DNA were being sequenced. In 1995, a team at The Institute for Genomic Research (TIGR) shotgun sequenced the entire genome of the bacteria *H. Influenza*¹⁹, for which $G \approx 1,800K$ bases. But while the source genome length has been increasing steadily, the average read length, L , has not been keeping pace. The analysis of Lander and Waterman allows us to examine how this is expected to effect the results of shotgun sequencing. It might be surprising to note that if the coverage, c , and the average read length L are held fixed, then three out of the four values listed above remain unaffected by the increase in source length. As an example, a couple of typical values in shotgun sequencing are $L = 500$ and $c = 10$. Using these numbers one would expect:

- $\approx 99.995\%$ of the source sequence to be covered
- Average contig length of $\approx 1,101\text{K}$ bases
- Average gap length of 50 bases

The percent of the source covered, the average contig length, and the average gap length remain constant. Only the expected number of gaps increases linearly with G . Recall the average number of gaps $g = Re^{-c} = (cGe^{-c})/L$. Substituting $L = 500$ and $c = 10$ we get $g \approx G/1,101,000$. So for $G < 1,101,000$ the expected number of gaps is less than 1. Notice that even for the H. Influenza, the expected number of gaps is only 1.6. Estimates for the size of the genome of *Drosophila* (the fruit fly), whose genome is currently being shotgun sequenced, is 130M bases of euchromatin DNA. Thus, using the values for L and c given above, one should expect approximately 118 gaps in the result. And if the entire human genome, which is estimated to be around 3.5 billion bases long, is sequenced using the shotgun method, then the expected number of gaps in the result is approximately 3,200.

4.4 Double-Barreled Shotgun Sequencing

Some of the issues of gaps in the solution can be addressed by *double-barreled shotgun sequencing*. This method begins the same as normal shotgun sequencing described above. The inserts are size-selected have length I , which should be larger than the average read length, L . The difference in the methods is that in double-barreled shotgun sequencing both ends of the inserts are sequenced²⁰. There are two possible results depending on the value chosen for I . If $I \leq 2 \times L - \Delta$, then the entire sequence of

the insert is known. The two reads obtained in this way can be combined into one larger sequence. This provides a method of increasing the effective size of L by almost a factor of 2 using currently available technology.

If $I > 2 \times L$, then the process above gives a pair of reads whose relative orientation and approximate distance from each other is known. These reads are called *mate pairs*. Let us define the coverage of the source DNA by the insert pieces to be $c' = R \times I / 2G$. Notice this is completely analogous to the definition of c since the total number of inserts is $R/2$, or half the number of reads. $I > 2L$ implies $c' > c$, so $g' = (Re^{-c'})/2$, the number of gaps between overlapping sections of inserts, is smaller than g , the number of gaps between contigs of reads. Recall that for *Drosophila*, we calculated $g \approx 118$. Now suppose $I = 2,000$, then $g' \approx 6.8 \times 10^{-12}$. And for human, where $g \approx 3,200$, $g' \approx 1.4 \times 10^{-10}$. This means that while there will be gaps in the solution, we do not expect gaps in the coverage by inserts. This information can be utilized to determine an ordering of the contigs.

Mate pairs that are placed in different contigs indicate a relative orientation of those contigs, and give an approximate distance between them. Define a graph whose vertices are the contigs. An edge connects two vertices if and only if the corresponding contigs contain mated reads. A subgraph that is a tree is called a *scaffold*. From the calculations above it is reasonable to hope that a single scaffold covers the entire source sequence and contains all the contigs. The ultimate goal is to use this information to obtain a complete ordering of the contigs. At this point a process known as Polymerase

Chain Reaction (PCR) can sequence the small gaps between adjacent contigs. PCR requires knowing 18-25 bases on either side of the gap to be filled.

One problem with relying on information from mate pairs is that they are unreliable. It is not uncommon to have 20% of reads that are reported to be mates actually come from unrelated parts of the source DNA. There are three major sources for this error. The first is that two fragments from different parts of the source sometimes get inserted into the same vector. These combined inserts are called *chimeric* clones and the end reads from them obviously come from unrelated portions of the genome. With careful size-selection and/or the use of asymmetric linkers in the insertion phase, this source of error can be kept as low as .01%. Another source of error is mistracking of a sample as it goes through the sequencing process. This is usually do to human error, such as a plate put in the wrong orientation or a sample being transferred to the wrong location. Using precautions such as dual-bar scanning any transfer and using asymmetric plates can keep the rate of false pairings due to this source to under .1%. The final and largest source of errors occurs in slab gel sequencing machines and is called the *lane-tracking problem*. Recall these machines sequence several reads simultaneously in the same block of gel by running them down separate lanes. The problem is the fragments don't travel through the gel in a perfectly straight line, but gently undulate as they progress, causing the optical scanning software to occasionally misnumber the lanes. This problem can cause 10% of the mate pairs to be erroneous. Capillary gel machines do not suffer from this problem since each read is run down a separate glass capillary.

4.5 The Human Genome Project

The Human Genome Project (HGP) is the largest public biology research project ever launched. It involves hundreds of labs, thousands of scientists, and billions of dollars of public funds. Its ultimate goal is the sequencing of all 3 billion bases (some estimates now put this as high as 4 billion) of the human genome. The large-scale public funding of genomics in the United States is an outgrowth of the Department of Energy (DOE) studies on the health effects of mutations. In 1986, David Smith publicly announced the DOE's intention to sequence the entire human genome. Though this proposal originally created great skepticism and opposition, by 1988 the National Institute of Health (NIH) and DOE had agreed to launch a joint program based on a common vision. The goal was to work in concert with the Sanger Centre in the United Kingdom and other labs in Europe and Japan to complete the sequence by 2005.²¹

The sequencing method being used by the HGP, often called the *clone-by-clone* approach, is a hierarchical two-tiered process. It is illustrated in Figure 4-6. The first half of the process proceeds very much like shotgun sequencing except for the scale. The source (the entire human genome) is first randomly fractured into many large pieces. These pieces are then size-selected to be between 50K and 300K bases long. Recall that when shotgun sequencing, the inserts are only 1K to 10K (for some double-barreled sequencing) bases long. As in shotgun sequencing, the next step is to clone the pieces. In this case, the pieces are too large to clone using phages, so *bacterial artificial chromosomes*, or BACs are used. For this reason, the pieces will be called BAC clones. Finally, a layout, called a *physical map*, of these clones is produced.

Since the sequences of the BAC clones are not known, the physical map (so named because it gives the physical location of the clone within the genome) cannot be determined by looking for matching sequences as is done in shotgun fragment assembly. Instead data characterizing the clones, called *fingerprints*, is collected and used to determine clone overlaps. Several types of fingerprint data has been defined and used to create physical maps. Some of these data types are:

- **Restriction Length Digests**: The approximate lengths of the pieces that result when a clone is cut at each occurrence of a particular substring of length 4, 6 or 8. The cutting is accomplished by using special proteins called *restriction enzymes*.²²
- **Restriction Maps**: The approximate locations along the clones of a selection of the particular substrings cut by restriction enzymes.²³

- **Oligo Probe Hybridization**: The presence or absence in the clone of each substring of length 12-24 within some set.²⁴
- **STS Probes**: The presence or absence of Sequence Tag Sites (STS), which are pairs of substrings of length 18 separated by a distance of 200-1000 bases in the clone.²⁵

STS probes are currently the most common type of fingerprints used. As with almost all experimental processes, they are prone to error. Probes are falsely reported to attach to clones (false positives) about 2-5% of the time and probes that are part of a clone are not reported (false negatives) about 10-20% of the time. Adding another degree of complication to the problem of assembling these clones is the fact that BAC clones are much more likely to be chimeric (a concatenation of disjoint pieces of the source) than are the smaller inserts cloned in phages. These factors make the construction of optimal physical maps computationally very difficult and heuristic methods are used.^{26 27} Worse, the sparsity of the information coupled with its mediocre reliability, creates much ambiguity in the answers delivered.

In the second phase of the process, the physical map is used to select a minimal tiling set of the clones. A tiling set is a subset that covers all clones in the physical map, which hopefully cover the entire genome. Each clone in the tiling set is then shotgun sequenced to obtain the sequence of the genome.

The HGP has been up and running for about 10 years and its prospects of at least partial success are good. Even though physical maps are difficult to build, they have been prepared for several chromosomes. Those are not complete, but they do cover a

significant portion of the chromosomes involved. Shotgun sequencing of BAC clones has become fairly routine. Reliable software is available, and machines to do the sequencing are being added at a rapid rate. The date for delivering the sequence of the entire genome, originally set at 2005, has been pushed up two years to 2003.

There are problems with the HGP strategy however. These primarily have to do with cost and efficiency, but there are also issues with the completeness of the result ultimately obtained. When designing a project of this magnitude, issues that should be considered are simplicity, the cost and ability to automate each step, and the scalability of the overall process. The HGP design, with its two separate processes of sequencing and physical mapping, does not handle these issues well. Sequencing can be highly automated once the insert library has been prepared, but under the HGP plan one will have to prepare a minimum of 25,000 clone libraries of BACs by hand. One will also have to continue to laboriously build and try to complete physical maps of each of the chromosomes. Originally it was anticipated that within 5 years of the project all the physical maps would be completed at a minimal cost. Eight years into the project there were maps of only a few chromosomes, most of which have on the order of hundreds of gaps with some of considerable size. The cost has also been much higher than anticipated. Another problem, as mentioned before, BAC clones can be chimeric. By some estimates 1-5% of the BAC clones being sequenced are actually composites of two or more unrelated pieces of the human genome.

4.6 Whole Genome Shotgun Sequencing and the Sequencing Race

It used to be widely accepted that cosmid-sized clones (a cosmid is yet another type of vector) of 30-50K were the upper limit on DNA molecules that could be successfully shotgun sequenced. When TIGR shotgun sequenced the entire *H. Influenza*, those perceptions were shattered. Now the ability to shotgun sequence BAC clones 200K bases long is taken as a given. These events led some to speculate on the possibility of shotgun sequencing the whole human genome. In 1997, James Weber and Eugene Myers published two papers advocating a whole-genome shotgun strategy for sequencing the human genome^{28,29}, and asked that the NIH fund a pilot project. Their proposal created a lot of controversy and the established community eventually rejected it³⁰.

The Weber and Myers proposal called for using dual-end sequencing inserts of two different sizes, termed *shorts* and *longs*. The short inserts would have a length of 0.4-1.2K bases. Thus the reads of the ends would overlap and would thus be combined into one long read as described in section 4.4. The long inserts would be 5-20K bases in size. They would be used to link contigs into scaffolds. They called for a ratio of long to short inserts that would be ≥ 1 .

Another key component in this version of whole-genome shotgun sequencing is the availability of pairs of unique marker sequences known to be 50K to 300K bases apart in the genome. For example, such long-range correlated pairs could come from current maps of STSs, or from a collection of BAC-end sequence pairs. BAC-end sequence pairs are reads from the ends of a BAC clone. Such marker pairs are available for the human genome now^{31, 32}. Thus reconstructing the target genome breaks down to

the problem of reconstructing the stretches of the genome between the marker pairs. This is referred to as the *inter-marker assembly problem*.

To initially demonstrate the feasibility of whole-genome shotgun sequencing, Myers and Weber ran computer simulations designed to determine if their approach would result in sufficient coverage and linkage. A simulated piece of DNA with two types of repeats was used. Short repeats were used to model the small, interspersed nucleic elements (SINEs) like Alu's. SINEs had a length of 300 bases and by default they covered 20% of the genome. Long, interspersed nucleic elements (LINEs) were modeled by repeats of length 1,500K bases that covered 5% of the genome by default. These repeats were positioned in a uniform random manner across the genome. To be conservative, the repeats were considered to be identical in sequence. Unique markers, which were 300 bases long, were positioned every 100K bases along the genome. Tests were then run varying the values of parameters such as coverage, read length, amount of repetitive DNA, ratio of long to short inserts, and *MIN_OVERLAP* to determine how often a *repeat-free scaffold* connecting adjacent markers existed. A scaffold is repeat free if every read included in its contigs has at least *MIN_OVERLAP* consecutive bases that overlap another read in the contig and those overlapping bases are not part of a repeat. It was found that for the default values of the parameters (these will be given later in this thesis) such a scaffold existed 99.8% of the time. This bodes well for the feasibility of the whole-genome shotgun approach.

The advantages of whole-genome shotgun sequencing over the clone-by-clone approach are listed below:

- **Simplicity:** This is the biggest advantage. Only *one* sequencing library needs to be built from the whole human DNA. After that, the remaining steps can be completely automated, thus the manpower required is low. There is no need to store BAC or other clones for any length of time. Furthermore, the system's simplicity allows it to handle some changes in the data characteristics with more ease. For example, it has recently been suggested that the human genome might be closer to 4 billion bases than the 3 billion currently estimated. If true, the HGP would have to re-evaluate their BAC libraries and physical maps. In contrast, a shotgun sequencing factor would just need to crank on the sequencers longer to obtain enough reads for the desired coverage.
- **Cost:** This is highly correlated to the previous advantage. Some believe that the simplicity of the process, coupled with the ease at which it can be automated, will allow whole-genome shotgun sequencing to be vastly less costly than the HGP approach. Weber and Myers estimated the cost to finish the sequence via the HGP approach to be 30 cents per finished base, and the cost of whole-genome shotgun sequencing to be 1 cent per raw base plus an additional \$90 million for informatics. This is a substantial difference.
- **No Physical Maps are Needed:** As described earlier, calculating physical maps is a difficult and inexact process.
- **Chimeric BAC Clones are Less of a Problem:** As stated earlier, BAC clones are much more likely to be composites than are smaller clones using different

vectors. These chimeric clones make physical maps more difficult to create, and, should such a clone end up in a tiling set, could result in an incorrect sequence being reported. If BAC ends are used as markers in whole-genome shotgun sequencing, then chimeric BAC clones mean some of these markers will be erroneous. This complicates the problem a little, but still has much less impact than on the HGP approach as the chimerism will be detected.

- Many Single Nucleotide Polymorphisms Will be Revealed: Everyone's DNA is not identical. This is what makes us individuals. A difference between two sequences from the same location in the genome is called a structural polymorphism. The most common difference between two sequences from the same location in the genome is a change to a single base. This is called a single nucleotide polymorphism, or SNP (pronounced 'snip'). SNPs are currently believed to exist at a rate of one every 1,000 bases along the genome. SNPs are of great interest since studying differences in genes is a powerful tool for figuring out gene function.

The 3-4 billion bases of the human genome are partitioned into 23 chromosomes. With the exception of the X-chromosome in males, every person has two copies of each chromosome, one coming from the father and the other from the mother. Thus, even if the DNA from a single person was used as the source for whole-genome shotgun sequencing, each chromosome would be represented twice, and many structural polymorphisms would be revealed. SNPs could be distinguished from sequencing errors by examining

the multiple alignment. Columns in which roughly half the reads indicate one base and the other half another (or no base at all) are probably due to SNPs. Columns where one read disagrees with the others are probably due to sequencing errors. If the DNA of several individuals is used as the source, this becomes more difficult. In this case one must use the quality numbers attached to the base call to make this determination. A base with a high quality number that differs from the other bases in the column might be due to a SNP. Since each BAC clone is from a single piece of DNA, the HGP will not reveal any SNPs except for where the sequenced BAC clones overlap.

- Avoidance of Redundant Sequencing: Since under the HGP scheme each BAC clone in the tiling set is shotgun sequenced separately, areas where these clones overlap will get sequenced twice. Thus these areas end up with twice the coverage needed. This double coverage will only occur over an estimated 10%-15% of the genome, but the time and cost it adds cannot be completely discounted.
- Speed: The relative simplicity of whole-genome shotgun sequencing not only means it will be more cost effective than the HGP plan, it should also be much faster.

The disadvantages to whole-genome shotgun sequencing are:

- Must Work With a Huge Data Set: Assuming the human genome is 3.5 billion bases long, then a total of 70 million reads of average length 500 must be collected to obtain a 10 fold coverage. To build an overlap graph would

require on the order of 4.9×10^{15} sequence comparisons. If there were no repeats, then the overlap graph would have 1.26 billion edges expected. Repeats will make this number much higher. These are daunting figures indeed.

- Massive Amounts of Repetitive Sequence: The human genome is replete with repeats. For example, Alu's and LINE elements are each thought to constitute 10% of the genome. It is estimated that 25% of the roughly 80,000 distinct genes believed contained in the human genome have two to five copies. There are also large duplications of 50-150K where a section of one chromosome has been copied onto another chromosome. A whole-genome shotgun approach must handle these repeats. This is its greatest challenge. The HGP approach on the other hand must only worry about repeats that cluster within a 50-200K BAC clone. This problem is significantly simpler.
- Unknown Results: While the HGP scheme has been in use for a while and researchers are confident in its results, whole-genome shotgun sequencing has not been tried out on higher organisms. The critics claim, with some justification, that the sequencing of *H. Influenza* does not demonstrate the feasibility of the whole-genome shotgun approach since the genome for *H. Influenza* does not contain the complex repeat structure of DNA from higher organisms.

In the last couple of years, the push to finish sequencing the human genome has become a bit of a race. In May of 1998 Craig Venter and the Perkin-Elmer Corporation

announced the start of new company, Celera Genomics, which would single-handedly sequence the entire human genome using the whole-genome shotgun technique³³. They expected it would cost \$250 million and would be completed by 2001, four years ahead of the HGP. A few months later, Eugene Myers took a leave of absence from the University of Arizona to lead the team at Celera building the software to assemble the genome.

Celera is utilizing Perkin-Elmer's new capillary gel sequencing machine, the ABI 3700 DNA Analyzer. The capillary gel technology eliminates the lane-tracking problem. The ABI 3700 has increased sensitivity and automation, and a larger throughput than any machine currently in use. The original plan called for 230 of these machines to be installed at Celera, which in operation would outstrip the capacity of all sequencing centers in the world combined. To process all the data, Celera entered into an alliance with Compaq-Digital to build one of the largest private computer facilities in the world. To test the feasibility of their approach, Celera will first attempt to shotgun sequence the genome of *Drosophila*, which is scheduled to be finished by winter of 1999.

The HGP reacted to Celera by announcing a new five-year plan in which they gave a new expected completion date of 2003, two years ahead of their previous schedule. They also announced that they would complete a "working draft" of the entire sequence by the end of 2001³⁴. In March of 1999 the HGP granted an additional \$81.6 million to three U.S. centers to scale up their sequencing efforts, and announced that they would now have a rough draft finished by spring 2000. Celera has also ramped up their efforts and now plan to use 300 ABI 3700's. Both sides claim not to be in a race with the

other, but are publicly skeptical of the other's chances for success. Whether they admit it or not, it is a race for the honor of being the first to make this landmark achievement in human history. No matter who is first, however, having the complete listing for the program of humans is sure to lead to some exciting changes in the years ahead.

4.7 What's in this Thesis

This thesis expands on the work done by Weber & Myers. In their paper they proposed reducing the problem of whole-genome shotgun sequencing to the inter-marker assembly problem. They demonstrated that for a variety of test cases, repeat-free scaffolds existed between adjacent markers with very high probability. In this thesis, we describe two algorithms for finding these scaffolds or *walks*, as they also will be called. The first algorithm attempts to form a walk connecting the markers as quickly as possible to verify their adjacency. The second algorithm finds a walk that reconstructs as much of the genome between the markers as possible.

We start by defining the data model we will be dealing with. This model includes a source sequence that contains repeats and markers scattered along it. The markers are unique sequence. That is, the repeats and markers do not overlap. The data model also includes a database of reads uniformly sampled from the source. These reads are of two types. Reads of type *long* have a mate pair, and reads of type *short* do not. These model end sequences for long and short inserts as described earlier. Some long inserts might be paired erroneously (i.e. mate pairs might come from unrelated parts of the source sequence). Our model has several simplifications over the true biological data. One of

these simplifications is that all repeats of the same type are identical. That is, each type of repeat has a fidelity of 100%. In some ways, this is a conservative assumption since it means that each occurrence of a repeat cannot be differentiated from another. While we do allow repeats to be clustered, we do not model tandem repeats of very small sequences.

Next, the most basic operation called a *database query* is described. A database query takes a sequence and returns all reads from the database that overlap with it. These overlaps may be true or induced by a repeat (see Figure 3-4). The database query also detects the repeat edges on the sequence. How these repeat edges can be detected is described in Section 4.3. Section 3 describes the simulator that was programmed to model this data and to simulate the results of the database query. The simulator allows us to adjust the values of many parameters including the number, type, and frequency of repeats, the ratio of read types, and the coverage.

We then define footprints and the basic operations to manipulate them. A footprint is a contiguous set of overlapping reads together with links to other footprints. The operation *take step* is given a footprint as input, and returns a new footprint linked to the footprint input. The operation *expand* increases the length of the footprint it is operating on. These operations are used to build the *quick scan* and *inter-marker assembly* algorithm. The goal of the quick scan algorithm is to quickly verify the adjacency of two markers by finding a linked set of footprints, called a walk, connecting the markers. The algorithm will attempt to use only footprints that come from unique portions of the source sequence. The goal of the inter-marker assembly algorithm is to

find an ordered, linked set of footprints covering as much of the unique portion of the source between two markers as possible. These algorithms were implemented using the simulator described in Section 5 and tested using various values for the simulation parameters. The results of these tests are given in Section 8. These results are very promising. For most values of the parameters the markers are successfully linked over 99% of the time and the walk created by the inter-marker assembler covers over 99.8% of the unique portion of the source.

Finally, to try to keep current with this quickly changing field, Section 9 discusses some of the aspects of the assembly effort at Celera Genomics. Recall that Celera Genomics is a recently formed company that plans to sequence the entire human genome using the shotgun method. We will start by discussing their model of inserts, which differs somewhat from that proposed in this thesis. We also briefly discuss the simulator they are using to create test data. The structures built by the first steps of the Celera assembler are then outlined. Finally we describe how these structures can be used to achieve the goals of our two proposed algorithms.

5 DATA STRUCTURES AND BASIC MOVES

5.1 The Problem

We will start by defining a simplification of the inter-marker assembly problem for whole-genome shotgun sequencing. Begin with a source sequence, G , over a four letter alphabet. The composition of most of this sequence is unknown, but is assumed to be random, with each letter occurring with equal probability. Exceptions to this randomness are subsequences called *repeats*, which occur multiple times throughout G . These repeats are divided into two classes, *ubiquitous*, which occur with great frequency, and *low copy number*, which appear a much smaller number of times. The ubiquitous repeats model genome structures such as ALU's and LINE's. The low copy number repeats are generally longer and model things like repeated genes. Also positioned along G are unique subsequences, 300 bases long, called *Markers*. These markers will usually be placed along G at intervals that are usually around 100K bases long.

Subsequences of G , called *inserts*, are randomly selected using a uniform distribution. The sequences from both ends of these inserts are known and called *reads*. We classify the inserts based on their size. We usually divide them into two types: *short* and *long*. The short inserts are sized so that their reads overlap by roughly 100 bases, effectively giving the sequence of the entire insert (around 700 – 1,000 bases). We will refer to this entire sequence as a single read of type short (as opposed to two overlapping reads, see Figure 5-1(a)). The reads of *long* inserts do not overlap. Thus each long insert produces two reads whose approximate distance and orientation is known. Reads from

the ends of long inserts will be called type *long reads*, and a pair of reads from opposite ends of the same long insert will be call *mate pairs* (see Figure 4-1(b)). The length of long inserts, the mix of short versus long inserts, and the number of inserts collected are parameters to the experiment. Typically we assume that enough reads have been collected so that the total number of bases in the input data set is 10 times the length of the target genome, requiring roughly 70 million reads in the case of the human genome of 3.5 billion bases. We refer to the data set of shotgun collected reads and their pairings as the *shotgun database*.

Given a pair of adjacent markers, we would ideally like to use the information in the database to determine the sequence between the markers. There are a few obstacles to this goal, however. First, even at 10x coverage, statistically there will be regions of the genome, called *gaps*, which are not covered by any read in the database. Second, repeats in G cause reads in the database that originate from disjoint locations in G to appear to overlap. Thus the goal of whole genome shotgun assembly is to determine the *contigs* of overlapping reads between the gaps and repeats and to order these contigs into *scaffolds*, or *walks*, using pairs of reads from inserts whose ends are on opposing sides of a gap or a repeat. In other words, our goal is to find a walk that joins the markers and in which the contigs are as large as possible.

Another obstacle that must be overcome when building these walks is that due to chimerism in the insert or tracking problems in the sequence pipeline, the distance and orientation for some mate pairs might be completely wrong (see Figure 5-1(c)). Thus one needs to be able to detect and avoid using such erroneous information. These errors typically occur at a rate of 10% false mates for conventional sequencers due primarily to lane tracking errors, and less than 1% for the newer capillary gel machines that do not have this problem.

5.2 Database Query

The basic building block of the algorithms described in this thesis is the *database query*, which, given a sequence S , returns all reads in the database that *overlap* that sequence. Due to sequencing errors, the database query will have to look for ϵ -overlaps as described in section 4.3. Recall that an ϵ -overlap between two sequences is an overlap where the edit distance between the overlapped sections is less than or equal to ϵ times the length of the overlap, which must be at least *MIN_OVERLAP*. The edit distance between two sequences is the minimum number of single character changes of the form insertion, deletion, or substitution, that must be performed on one sequence to make it identical to the other sequence. The problem of finding ϵ -overlaps has been well studied and satisfactory solutions for it exist⁹, so the implementation of this aspect of the database query will not be elaborated here.

In addition to returning the reads that overlap the queried sequence, a key piece of information returned by the database query is any *repeat edges* detected on the queried sequence. The queried sequence is always a subsequence of the source sequence, usually a subsequence of some read, which we will call a *subread*. Note that as illustrated in Figure 5-2, it often happens that a significant portion of the queried sequence and a read in the database can match due to a repeat, but then fail to continue to match as one moves into the unique regions flanking the two repeat copies inducing the matching segment. We have found, in trials on real sequences, that it is possible to detect such “chimeric” alignments and to pinpoint within a few bases where the repeat edge occurs. Doing so is an exercise in sequence comparison, and the process will be explained in further detail in

section 5.3. We assume that if a sequence has such a detectable edge (at least *MIN_OVERLAP* bases from either of its ends), then the database query engine will detect and report it.

There are many cases where because of ubiquitous repeats the number of reads overlapping a sequence is so large as to be uninformative. Finding and reporting all these overlaps is a waste of both time and space. We thus assume that when the database query finds *MAX_OVERLAPS* with the queried sequence, it terminates its search of the database and returns an indication that too many overlaps have been detected. It does still return the repeat edges that it detects, however.

5.3 Repeat Edge Detection

Suppose we have the situation as illustrated in Figure 5-2 where two reads overlap the right side of different occurrences of the same repeat type. Let **M** be the dynamic programming matrix that gives the edit distance between overlapping sections of the two

sequences. That is, the score for an insertion, deletion, or mismatch is 1. We can tell that there is a repeat edge to detect by searching the band made up of the row number Δ and column number Δ for a small entry. (Recall Δ is shorthand for *MIN_OVERLAP*.) More precisely, check to see if there exists a matrix entry $m_{ij} \leq \epsilon \times \Delta$ in the band of entries with $ij \geq \Delta$ and $i = \Delta$ or $j = \Delta$. (See Figure 4-3) This indicates a significant match between one sequence and the start of the other. Suppose m' is the smallest entry in the band. In Figure 4-3 m' is shown as a star on the band. If $\mathbf{M}$ has a sufficiently small value on its last row or column, then an ϵ -overlap exists between the sequences. That is, suppose the dimensions of $\mathbf{M}$ are $N \times K$, then there is an ϵ -overlap between the sequences if and only if there exists an $m_{ij} \leq \epsilon \times \min(i, j)$ such that $ij \geq \Delta$ and $i = N$ or $j = K$. Since such an overlap does not exist in our example, somewhere along the path that defines the best alignment the scores must begin rising quickly (*i.e.* the alignment becomes very poor). Our goal is to locate this point.

Notice that in Figure 4-3 the point we seek is labeled as the minimum value on the path. This certainly is not true when using the edit score for creating the matrix since it is nondecreasing along the path. Instead we would like to have a scoring scheme that is negatively biased for the portion of the overlap corresponding to the repeat and positively biased for the rest. This would mean that the score along the path would decrease while it corresponded to overlaps within the repeat, and increase outside the repeat. Thus the edge of the repeat would be indicated by the minimum value on the path as desired.

To define the desired scoring function, we introduce a factor δ that is subtracted from the standard edit distance score for each horizontal or diagonal move on the path, with the exception of the top row. In other words, the score for any entry is the edit score minus δ times the length of the portion of the queried fragment being aligned. More precisely, let $s = s_1 s_2 \dots s_n$ be the queried sequence and $r = r_1 r_2 \dots r_m$ be the read being compared to it, then the formula for calculating an entry in the dynamic programming matrix M is

$$m_{i,j} = \min \begin{cases} m_{i-1,j} + 1 \\ m_{i,j-1} - \delta + 1 \\ m_{i-1,j-1} - \delta + d(r_i, s_j) \end{cases}$$

where

$$d(r_i, s_j) = \begin{cases} 0 & \text{if } r_i = s_j \\ 1 & \text{otherwise} \end{cases}$$

To choose the correct value to use for δ , we must determine what the expected value for the edit distance per length is for the cases when the sections being aligned are from a repeat and when they are from unrelated parts of the source sequence. For the case of the sequences coming from different occurrences of the same repeat, this value is by definition one minus the fidelity of the repeat. That is, if the repeats are 90% similar, then .1 is the desired value of δ . It would be nice if the value for the case of random sequences could be calculated exactly, however this is a surprisingly difficult problem and it remains unsolved. So instead we ran some empirical tests to determine this value.

We start with two sequences S and R with $|S| = K$ and $|R| = 2K$. The composition of both S and R is random with each of the four characters being equally likely. Next an alignment is done between the two sequences, starting at the first character of each, and ending when all characters of S have been aligned. This is to simulate the alignment, starting at the point beyond the repeat edge, of a queried sequence and a read, which may be longer than it. We are interested in the alignment score divided by K . Clearly, if the two sequences were lined up and the differences counted, the expected value would be .75 (since one out of four bases on average would match). Allowing alignment reduces this value significantly. The results of the tests are shown in Table 1. Notice that the value decreases as K increases. This is because a longer sequence allows for more ‘play’ in the alignment. This value does appear to be approaching a limit, though the tests stopped at $K = 500$ since that is the expected length of a read.

Length (K)	50	100	150	200	250	300	350	400	450	500
(Alignment score)/ K	.511	.495	.488	.485	.482	.480	.478	.477	.476	.475

Table 1: Alignment score per length for random sequences

The value chosen for δ should lie between .475, and the highest fidelity of repeats that might cause false overlaps. This last number will depend on the sequencing error expected in the reads. If the sequencing error is low enough that the reads are at all useful, repeats with fidelity of less than 80% should not cause complications. Thus we can choose a number between .2 and .475 to be δ . We used the value of .35 and ran multiple tests to see if edge of repeats could be detected. These tests consisted of generating a random source sequence that contained repeats. This was uniformly broken into fragments. Some of these fragments were compared against the other fragments to search for repeat edges. If a fragment was found to contain a repeat edge, the location of that edge reported was an average of all the locations determined by the pair-wise comparisons. The results were very accurate. No repeat edges were missed, and their locations were correct to within a couple of bases. While the scope of these tests was limited (a 75K source broken into 1,500 fragments), they do indicate that a repeat edge detection scheme based on this scoring function can be successful.

6 THE SIMULATOR

To test the effectiveness of our algorithms we built a simulator that allows us to try out many variations of our algorithm within a manageable time. The simulator must model the database of reads from a whole human genome in such a way that the overlaps between the reads can be determined quickly. Memory conservation was also an issue in the design of the simulator. These concerns had to be balanced against the need to accurately simulate the characteristics of the genome.

6.1 What is Simulated

The simulator uses a number of variables to define the nature of the modeled genome. One can set the size of the genome, *GENOME_LEN*, in bases, and the number of types of repeats, *NUM_CLASSES*, contained in the genome. For each type $i \in [1, NUM_CLASSES]$ of repeat, the parameters, *REPEAT_LEN*[*i*], *REPEAT_PERCENTAGE*[*i*], *CLUSTER_PROB*[*i*], and *CLUSTER_DENSITY*[*i*], control the repeat type's size, frequency, and clustering characteristics.

Parameter *REPEAT_LEN*[*i*] specifies the length of repeats of type *i*, and *REPEAT_PERCENTAGE*[*i*] gives the percent of the genome covered by repeats of type *i*. Thus the number of occurrences of repeat *i* is

$$(GENOME_LEN \times REPEAT_PERCENTAGE[i]) / REPEAT_LEN[i]$$

The simulator makes the conservative assumption that every occurrence of a repeat of the same type is identical in sequence.

Repeat occurrences of the same type may be clustered into groups whose cardinality is geometrically distributed. Each accumulating cluster is extended to the next repeat instance with probability $CLUSTER_PROB[i]$ for an average cluster size of $1/(1 - CLUSTER_PROB[i])$. Notice that if $CLUSTER_PROB[i] = 0$ then each cluster contains only one repeat occurrence (i.e. no clustering). Each multiple repeat cluster has all of its, say r , repeats uniformly distributed within a segment of the genome of size $r \times CLUSTER_DENSITY[i] \times REPEAT[i]$. The parameter $CLUSTER_DENSITY[i]$ must be at least 1, corresponding to the case of tandem repeats. These cluster segments, together with the non-clustered repeat occurrences, are distributed uniformly across the genome without overlap.

The length of genomic markers is controlled by the variable $MARKER_LEN$. The distance between markers is determined by the variables $MARKER_SPACING$ and $MARKER_VAR$, which give the average spacing and a percentage of variation. Specifically, for every pair of adjacent markers the inter-marker distance is uniformly chosen from the interval $MARKER_SPACING \times (1 \pm MARKER_VAR)$. Each marker is expected to be unique, so we always place them so they do not overlap with any repeats.

The simulator models a shotgun database of reads of types long and short, as described earlier. The mean lengths of the inserts are determined by the variables $SHORT_LEN$ and $LONG_LEN$, respectively, and the inserts have length uniformly chosen to be $(1 \pm LEN_VARIATION)$ times the mean length. The ratio of long to short inserts collected is controlled by the variable LS_RATIO . Each sequence read is assumed to be of length $READ_LEN$. Recall that the end-reads of long inserts do not overlap. For

short fragments we assume the reads overlap giving a sequence of length equal to that of the insert. The simulator models a total number of inserts to give a base count equal to $COVERAGE \times GENOME_LEN$ where $COVERAGE$ specifies the desired sequence coverage of the genome. To be precise the number of short inserts generated is:

$$S = \frac{GENOME_LEN \times COVERAGE}{SHORT_LEN + LS_RATIO \times 2 \times READ_LEN}$$

And the number of long inserts generated is:

$$S \times LS_RATIO$$

The inserts generated are sampled uniformly from the genome. Lastly, it is essential to model the situation where two long reads are erroneously paired (i.e. they don't actually come from opposite ends of the same insert). The variable $LONG_ERR$ specifies the probability a given set of mate pairs is erroneous. When the simulator generates an erroneous mate pair, the two reads are taken from independent, uniformly chosen locations within the genome.

6.2 Simulator Primitives and Repeat Edge Detection

The basic action of the simulator is to answer the database query described in section 5.2. The simulator rapidly answers this question for a simulated genome in a manner that is a conservative approximation of what a true similarity-based search engine would produce. Note that a true implementation of such an overlap search against 35 billion characters is a time intensive operation and in the real scenario one would wish to minimize the number of such queries. Since the simulator knows where every read and

every repeat of every type is in the genomes it constructs, it can very efficiently simulate the results of a real database query.

Intuitively, the simulator considers the queried sequence to overlap a read in the database if their sequences would overlap under the assumption that all repeats of the same type are identical in sequence. Formally, suppose the queried sequence is an interval $[s,t]$ of the genome, and the read in question was sampled from the interval $[i,j]$. Recall Δ is shorthand for the simulation parameter *MIN_OVERLAP* that specifies the smallest overlap that one would consider statistically rare in the whole genome context, typically 35 bases if the sequence error rate is 5% or less. Then the simulator reports a true overlap between the sequences if and only if:

$$|[s,t] \cap [i,j]| \geq \Delta$$

Suppose there exists two occurrences of a repeat of the same type with locations on the genome of $[A,B]$ and $[C,D]$, then the simulator will report an overlap due to a repeat if:

$$\begin{aligned} & t \in [A, B + \Delta] \quad \text{AND} \quad i \in (C - \Delta, D] \quad \text{AND} \\ & \left| \left(([A,B] \cap [s,t]) - A \right) \cap \left(([C,D] \cap [i,j]) - C \right) \right| \geq \Delta \\ & \text{where by definition: } [x,y] - z = [x - z, y - z] \end{aligned}$$

The above defines the case where the read overlaps the right side of the queried sequence due to a repeat. The case where the read overlaps the left side is defined by the same expressions with the roles of $[i,j]$ and $[s,t]$ reversed.

In other words, the queried sequence and the read overlap if (1) they were sampled from overlapping segments of the genome of sufficient length, or (2) they share a repeat-induced overlapping segment involving less than Δ bases of the flanking

sequence to each side of the repeats. Our definition of overlap is thus very conservative in that in almost all instances it will report more overlaps than would actually be found by a routine that compares sequences.

Now suppose as before that the queried sequence comes from the interval $[s,t]$ of the genome and there exists a repeat at location $[A,B]$ such that $B \in [s + \Delta, t - \Delta]$ then the simulator will report the right edge of the repeat iff there exists a *different* occurrence of the repeat at location $[C,D]$ and a read from location $[i,j]$ such that $D \in [i + \Delta, j - \Delta]$. In other words it takes at least $MIN_OVERLAP$ bases on both sides of the repeat edge to detect it. The rules for detecting the left edge of a repeat are analogous. If the repeats are fairly short and closely clustered together, it is possible for a queried sequence to contain several repeat edges. In this case, rather than return a list of all repeat edges detected, it was sufficient for our algorithms if at most four different edges are reported. These include the leftmost right edge, the rightmost left edge, the rightmost *significant* right edge, and the leftmost *significant* left edge. By a significant edge we mean one in which there is at least $MIN_OVERLAP$ bases of unique sequence between it and the next repeat edge detected. For example, suppose that a the right edge of a repeat was discovered at location B and the next edge (a left edge) discovered along the sequence is at location C , then the edge at B is significant if and only if $C - B \geq \Delta$.

As specified in the definition of the database query, the overlaps reported never exceeds $MAX_OVERLAPS$. Once those many are detected, the count of overlaps is set to $MAX_OVERLAPS$, and the search for overlaps ends. The simulation represents a conservative estimation of what should really happen provided our repeat model of the

genome is sufficiently realistic. The results given in Section 8 show that simply being able to detect the edges of repeats suffices to assemble the unique portions of the genome with high confidence.

7 THE QUICK SCAN ALGORITHM

The object of the *quick scan* algorithm is to quickly confirm the adjacency of two markers by finding a repeat-free scaffold, or *walk*, connecting them. Since querying the database will be by far the most time intensive operation, we measure the speed of the algorithm by the number of database queries it generates. Putting a limit on the number of database queries allowed, we determine how frequently our algorithm succeeds in finding a connecting walk between adjacent markers.

7.1 Basic Ideas

The quick scan algorithm searches for a connecting scaffold between two markers that are believed to be adjacent. Since we visualize this process as starting at one marker and walking across the genome in search of the other, we will call the desired scaffold a *walk* and each contig in the walk will be called a *footprint*. Thus we imagine traversing the genome by stepping from footprint to footprint via mate pairs that have one read in each footprint being stepped across. Since we know the approximate distance between mate pairs, we know approximately how far we've walked. However, the variance in the distance between markers tends to be fairly high, and the marker is a relatively small sequence. Thus, to increase the chances of connecting the markers quickly, we explore walks from each of the two markers simultaneously in an attempt to meet somewhere in the middle. See Figure 7-1 for an example of the anatomy of a walk.

As we walk across the genome, there are a couple of events that can seriously throw us off our course. First, taking a step based on an erroneous mate pair can cause the walk to end up at some random place in the genome. We avoid this by only taking *confirmed* steps. This will be defined precisely later, but basically we require evidence from two separate sets of mate pairs before we accept a mating as true. The second problem spot is repeats in the genome. Because repeats are identical in sequence, a contig that consists of reads sampled from inside a repeat will contain all overlapping reads from every occurrence of that repeat. Thus a mate pair used to take a step out of the repeat may come from any occurrence of the repeat. Using the analogy of walking along the genome, a repeat is like a teleporter into which you step at one location, and step out at some other location along the genome (Figure 7-2). The obvious solution to this is to avoid stepping into a repeat if possible. Methods to achieve this are outlined in

the next few sections where we define a footprint and the basic operations used on it. We then detail how these operations are put together to create the walk.

7.2 The Footprint

The basic building block of our walk is a *footprint*. A footprint is a contig, so it contains a list of overlapping reads together with their relative positions. However, a footprint also contains information needed for the walk. This information includes the direction of the walk that the footprint is part of and the estimated distance of the footprint from the marker at which the walk began. In addition, all known repeat edges that overlap the footprint are kept track of. A footprint also contains a set of right and left links to adjacent footprints. Two footprints will be considered to be adjacent (connected by a link or *stride*) if and only if they contain at least two sets of corresponding mate pairs. Requiring two sets of mate pairs dramatically reduces the probability that two

footprints will be erroneously linked since this would require two false mate pairs that were sampled from correlated parts of the genome.

To facilitate the process of taking a step, for each footprint we keep a list of all the reads that have a mate pair in the footprint and are in the direction of the walk. These reads are places that can possibly be stepped to from the footprint, and so we will call this the set of *possible steps*. Throughout the course of the walk some of these reads will be determined to come from portions of the genome that contain repeats. For each of these reads we keep track of a subsequence that is not known to be repetitive. These subsequences are compared against each other to search for overlaps. If one of these subsequences is not at least *MIN_OVERLAP* in length, then the read that contains that subsequence is marked as *REPETITIVE*. *REPETITIVE* reads are not used in these comparisons. Reads for which such an overlap is found are labeled as *confirmed*.

While a footprint contains various other bookkeeping information which will be described as needed, we have now given a sufficient description to be able to detail the basic operations of the quick scan algorithm: *take step* and *expand* a footprint.

7.3 The Take Step Procedure

The basic operation *take step* starts with a footprint and takes a step in the direction of the walk (via a mate pair), thus creating a new footprint. As discussed in section 7.1, a bad step can be taken two ways. First, a step can be taken using mates that are erroneously paired. We avoid this pitfall by only using confirmed possible steps as described in the previous section. The other source of error is stepping into and out of a

repeat. As stated earlier, the way to avoid this error is to attempt not to step into a repeat. To do this we need to know when a repeat has been stepped into. There are three methods for detecting that a repeat has been entered. These methods are outlined below.

The first method for detecting a repeat has been stepped into is checking to see if the database query finds a repeat edge. To create the new footprint, the *take step* routine takes a confirmed possible step, say r , and queries the database with its subsequence that is not known to be repetitive. For convenience, we will denote this subsequence as r' . If this database query finds a repeat edge on r' , then r' is no longer a subsequence that is not known to be repetitive. We use the repeat edge information to trim r' so that it only contains a subsequence not known to be repetitive. Note that this process might reveal all of r to be repetitive (i.e. r' is now empty). Since we don't want to use repetitive portions of the reads to confirm possible steps, we must update the status (confirmed or not confirmed) of the possible next steps using the new version of r' . If r remains confirmed, another attempt to step to it can be attempted.

The second sign that a repeat has been stepped into is the number of overlaps returned by the database query. The value of *MAX_OVERLAPS* is chosen to be high enough that there is no chance it will be reached due to real overlaps. Therefore if a database query finds *MAX_OVERLAPS* or more overlaps, then it is safe to say that r' contains a repeat. If no repeat edge was detected then we must assume all of r is repetitive. This method allows us to catch reads that lie within a ubiquitous repeat, but may fail to catch reads that lie completely within a low copy number repeat.

Figure 7-3: Detecting that r' is in a repeat by inconsistency with s and t .

Lastly, even if r' lies completely within a low copy number repeat, the repeat might still be detected if r is sufficiently close to the edge of the repeat. Let S be the set of reads returned by the database query of r' . We shall say S is inconsistent if the overlaps that are transitively implied do not exist. For example, if s and t are elements of S that overlap the right side of r' , then there is an implied overlap between s and t . If r' lies within a repeat but close to its edge, then it is possible that s and t straddle the repeat edge of different occurrences of the repeat, and therefore do not overlap (see Figure 7-3). By choosing the order correctly, S can be checked for consistency by doing one sequence comparison for each element of S . With these preliminaries, we can now detail the *take step* routine as follows:

Procedure Take_step (f : footprint)

- Find the read r from f 's list of confirmed possible steps that lies furthest from f and is not part of another footprint. In no such read exists, return *FAILED*.

- Query the database using the subread of r not known to contain a repeat. Let S be the set of overlapping reads returned by the query.
- **If** a repeat edge is detected on the subread **or** too many overlaps are detected **or** S is inconsistent **then**
 - Update the repetitive segment information for r , which is all of r except when a edge is detected.
 - Update the status (confirmed or not confirmed) of f 's possible list of possible next steps
 - Restart the Take_step routine
- The new footprint $g = S \cup \{r\}$ is recorded as having f as its parent (this is used for backtracking)
- Find and record all links from g to existing footprints.
- **If** g has reads overlapping an existing footprint **then** merge the two footprints into a single one.

As outlined a jump step has one of the following outcomes: (1) no jump is possible or (2) a new footprint is returned and either (a) found to merge with another, or (b) remains disjoint and is the current last point of the walk

7.4 Expand a Footprint

The other basic action on footprints is an *expansion* in a given direction. The goal is to extend the set of overlapping reads that make up the footprint, thus covering more of

the genome and increasing the size of the set of possible next steps. This is accomplished by doing a database query using a sequence from the end of the footprint. The footprint is then updated with the returned reads.

There are two reasons why it may not be possible to expand a footprint. First, the footprint may abut a gap in the coverage. Note that by a gap here we don't necessarily mean that there is no read covering the genome next to the edge of the footprint, but that there is no read that overlaps the edge of the footprint by at least *MIN_OVERLAP* bases. A gap cannot be expanded over, but as we expect the gaps to be small, we should be able to step over them.

The second impediment to an expansion is a repeat. If the queried sequence overlaps an ubiquitous repeat, then there will be too many reads overlapping it, and the database query will not return them all. So even if we wanted to include them in the footprint it would not be possible. If we did include reads internal to a repeat as part of a footprint, then those reads could come from any occurrence of the repeat along the genome. Thus using mates of those reads as possible next steps could result in a bad step exactly as described in the previous section. Lastly, if the expansion made it to the opposite end of the repeat, the set of reads *S* returned by the data base query might be inconsistent, and we would not know which reads from *S* should be added to the footprint. For example, in Figure 6-3 suppose *r'* comes from the end of a footprint, then it would be unknown whether *s* or *t* should be used in the expansion. For these reasons we do not want to include reads internal to a repeat within the footprint.

We rely heavily on the ability to detect repeat edges to deal with repeats during the expansion operation. For example, suppose we wish to expand the footprint f to the right, and further suppose, as shown in Figure 6-4, that the rightmost read, r in f straddles a repeat. A database query on r will return the true overlaps on r , but also those overlaps that are caused by the repeat. The database query will also give the edge of the repeat on r , and thus we know those reads that overlap the unique portion of r are true overlaps. If the repeat is ubiquitous, then all those overlaps will not have been returned, and the database must be queried again, this time using only the nonrepetitive portion of r . (In the figure this is shown as the thin part of r .) This has two major benefits. First, by adding reads that truly overlap f we increase the size of the set of possible next steps, and thus have a better chance of taking a confirmed step from f . Second, if the repeat is fairly short (e.g. an ALU), there might exist a read in the database that spans it. That is, a read, such as that labeled s in the figure, with enough unique sequence on the left so that we know it should be added to the footprint, and enough unique sequence on the right so that the footprint can be expanded further in that direction. Note that we will not know that s spans the repeat until we query the database with s and discover the right edge of the repeat.

It is also possible that the footprint being expanded might lie within a low copy number repeat. In such a case, the expansion might reveal that fact in one of three ways. First, the database query might find at least *MAX_OVERLAPS* overlaps. This can happen if the frequency of the repeat is such that queries of some subsequences of the repeat return just under *MAX_OVERLAPS* overlaps and others just over. This would allow the

footprint to be created by querying a read from a portion of the repeat that by chance has been sampled less than other parts of the repeat. Second, a repeat edge could indicate that the footprint lies within a repeat. For example, if a footprint being expanded to the right reveals the right edge of a repeat and there is no known left edge in the footprint, then the footprint must lie within some large, low copy number repeat. (This assumes that edges are not missed.) Lastly, as before, if the set that is returned by the database query is inconsistent, then the sequence queried lies within a repeat.

Suppose we are expanding a footprint in direction d , then we will say that a repeat edge is a *exterior* edge if it indicates the repeat is being expanded into. For example, if a footprint is being expanded to the right, then the left edge of a repeat is the exterior edge. We will call the other edge the *interior* edge. We are now able to outline the algorithm for *expand*:

Procedure *Expand* (*f*: footprint, *direction* {left, right})

- Query the database with a subsequence *s* of the end of *f* that is in the desired *direction* and assume *S* is returned
- **If** the side of *f* being expended is known to lie in a repeat (i.e. an exterior edge has been previously discovered) **then**
 - **If** a interior edge was discovered **then**
 - query again using the portion of *s* outside the repeat edge
 - **Else**
 - return *FAILED* (the repeat can not be spanned)
- **Else** (There is no known repeat.)
 - **If** the exterior edge of a repeat was discovered **then**
 - Query again using the portion *s* that lies outside the repeat
 - **Else if** a interior edges of a repeat was discovered **or** *S* is inconsistent **or** too many overlaps were detected
 - Mark footprint as *REPETITIVE* and return *FAILED*
- **If** *S-f* is empty **then** return *FAILED*
- Add the reads in *S* to the footprint, updating the links and set of possible steps as necessary.
- **If** *f* has reads overlapping in an existing footprint **then** merge the two footprints into a single one.

Choosing the sequence *s* to query the database with requires a little care. One might be tempted to use the read that lies furthest in the direction of the expansion, and

indeed this was done with early implementations of the algorithm. This causes a couple of problems, however. First, for very small footprints, using a database query on the furthest read, r , might result in a set S that expands the footprint in both directions. This would violate the definition of *expand* and requires a lot of extra code to handle this special case. For example, whether or not a repeat edge is exterior or interior would now depend on where on r the edge as well as which side of the edge the repeat lies. To handle this issue for each footprint we will keep track of the right and left-most points on the footprint on which a database query has been performed. We will call these points le and re respectively. When the footprint is created, these points are set to the ends of r' , the subread used to create the footprint as outlined in the *take step* operation. When the footprint is expanded in the direction d , the corresponding point (le or re) is set to the outer edge of s , the subsequence used to query the database as outlined above. Thus, when doing an expansion, it would be redundant to include the sequence between le and re in the database query. This indicates we only need to query on the portion of r that lies outside these points.

However, there is a second problem. Suppose a footprint f is expanded to the right by doing a database query on its furthest read r . This will cause re to be set to the end of r . Suppose that this expansion results in the read r' , which overlaps the last $MIN_OVERLAP$ bases of r' , being added to the f . Suppose further that r' is now the read in f furthest to the right. Lastly, suppose there is a repeat that overlaps the last $MIN_OVERLAP - 1$ bases of r . Then the edge of this repeat will not be revealed by the database query of r , since the number of bases is too small. If another expansion to the

right of f is attempted by using r' to query the database, the exterior edge of the repeat will not be revealed (since r' has only one base to the left of the repeat). Thus we will have unknowingly expanded into a repeat. To insure that the repeat edge is detected in this situation, we need to include the last $2 \times \Delta - 1$ bases of r in the second database query. Thus we can define s in the take step operation as the subsequence of f that runs from $re - (2 \times \Delta - 1)$ to the rightmost edge for expansions to the right and from $le + (2 \times \Delta - 1)$ to the leftmost edge for expansions to the left.

7.5 The Quick Scan Search

The quick scan algorithm begins by doing a database query on each marker to create two initial *active* footprints. Each walk will have a footprint designated as the active footprint. This is just the footprint currently being searched, or, in terms of our analogy, the current location of the walker. The algorithm starts by simultaneously walking from both directions, trying to quickly traverse the distance between the markers. Recall each footprint contains an estimate of its distance from the starting marker of the walk. This is only an estimate because of the variation in the lengths of the long inserts. We define a value, *bound*, which is a target distance for each walk to travel. *Bound* should be large enough that if each walk travels that far, then the combined *actual* distance traveled by each walk is larger than the actual distance between the markers (assuming the markers are really adjacent). The value for *bound* used by the final implementation of the algorithm was *MARKER_SPACING*, the average distance between markers.

While the estimated distance of the active footprint is less than *bound*, each walk takes turns doing one of the actions in the prioritized list below. When one of the walks has traveled far enough, or has exhausted all possibilities, the other walk continues to make moves from the list until it terminates as well. There are two events that will terminate the algorithm completely. One is if the two walks meet. This can happen either because footprints from the two walks become linked, or if an *expansion* or *take step* move results in footprints from the two walks being merged. If the walks meet, then quick scan is successful. Since the goal of quick scan is to *quickly* connect the markers with a walk, we put a limit on the number of database queries that can be issued. At any point in the algorithm when that limit is reached, the algorithm has failed and terminates. With that understood, here is the list of moves the walks alternate making as they attempt to cross the genome:

1. Try to take a step from the active footprint. If that succeeds the new footprint becomes the active footprint. If not:
2. Try to expand the active footprint in the direction of the walk. If this is successful then there are more possible steps for the next move. If not:
3. Try to expand the active footprint in the opposite direction of the walk. If this fails then:
4. Backtrack to the parent of the active footprint if possible (i.e. the parent of the active footprint becomes the active footprint). If this is not possible (the active footprint was the initial footprint), then all possibilities have been exhausted and the walk in this direction terminates.

It is possible that at the end of this phase of the algorithm the walks have not met, one or both of the walks traveled the required distance, and the quota of database queries has not been exhausted. At this juncture one can expand and/or continue to step from the scaffold footprints until the two walks meet, if at all. There are several variations how to prioritize the moves in this second phase of the scan and we tried several different schemes to see which is fastest. Each scheme is iterated until the walks meet, exhaustion occurs, or the database query threshold is exceeded.

1. For each footprint, do an expansion if possible, otherwise take a step.
2. For each footprint, take a step if possible, otherwise do an expansion.
3. Alternate between 1. and 2.
4. For each footprint, expand until a step is possible and then take the step (if possible).
5. For each footprint, expand as much as possible and then take all steps.

In the first three protocols above, only one action is taken on each footprint per iteration. In the last approach all actions that can be taken for a footprint are, so each footprint is only acted on once. Expansion on a footprint is always done first in the direction of the walk if possible. If expansion is no longer possible in that direction, it is then done in the other direction. Since it is a wasted effort to build footprints beyond the target marker (there is no chance of meeting the other walk in that area), no expansion is done in the direction of the walk on a footprint that has reached the distance *bound* and no footprint whose estimated distance is beyond *bound* is stepped to.

COVERAGE					
STRATEGY		10	9	8	7
	1	83.4	87.7	76.6	114.0
	2	91.9	106.4	81.2	120.6
	3	94.2	110.1	80.7	126.1
	4	81.4	85.9	72.0	111.0
	5	51.7	58.4	56.4	92.8

Table 2: Average database queries for the different meeting strategies

The results our tests of the different meeting strategies are given in Table 2. These tests were run using our simulator with the default values of the parameters given in Section 8. For each of the meeting strategies we counted the number of database queries required by the quick scan algorithm to connect adjacent markers. The average number of database queries used over a large number of such tests is given in Table 2. These results show a clear win for approach 5, and the reason is simply that in almost all cases the footprints of each walk have landed in all the unique stretches between the markers, so expanding anyone of them will meet a footprint from the other walk that is also in that unique region. We also ran tests to see if there was a best order in which to examine the footprints for this particular strategy. As expected these did not result in large differences, but there was a slight advantage to examining the footprints in the middle of the walks first and then working our way to the outside.

One must observe carefully that the quick scan algorithm is attempting to find a repeat-free scaffold. Note that the *take step* operation avoids ubiquitous repeats easily as the database query on the possible step r' will always report too many overlaps for such a region. But low copy number repeats can easily be stepped into without setting off an obvious alarm. Note that the *expand* operation is careful to notice if a repeat edge detection signals that a footprint was in a repeat and marks it as *REPETITIVE* so that it will no longer be considered. In order to avoid footprints in low copy number repeats, we further classify footprints as *UNIQUE*, *SUSPICIOUS*, or *UNKNOWN*. Footprints that are known to be unique because the exterior edge of a repeat has been discovered are classified as *UNIQUE*. Footprints suspected of being repetitive because the coverage depth for the footprint is greater than the expected coverage by a certain factor (we experimentally determined that $1.5 \times \text{COVERAGE}$ works well) are classified as *SUSPICIOUS*. Lastly, if nothing has yet been detected about the footprint, we classify it as *UNKNOWN*. The quick scan algorithm always gives precedence to *UNIQUE* and *UNKNOWN* footprints in whatever move sequence it is making.

As a final note, one should observe that the quick scan algorithm is guaranteed to fail to find a repeat-free walk whenever there is a repeat whose length approaches or exceeds *LONG_LEN* between the markers. We did not develop strategies to rectify this, as the idea of the quick scan algorithm is to *quickly* prescreen for valid marker pairs without unusually large repeats between them. After these “easy stretches” have been assembled and their associated reads removed from the database, greatly reducing its

size, then and only then do we utilize the inter-marker assembly algorithm below to assemble regions of higher repeat complexity.

8 THE INTER-MARKER ASSEMBLY ALGORITHM

The object of the *inter-marker assembly* algorithm is to fill in as much of the genome as possible between two adjacent markers. The desired output of this algorithm is therefore a set of ordered, maximal contigs (or footprints). The different goals for the inter-marker assembly algorithm and the quick scan algorithm dictate a differing order of actions. While the quick scan preferred stepping rather than expanding in order to quickly find a walk between the markers, the inter-marker algorithm prefers expanding over stepping in order to fill in the sequence as completely and safely as possible.

Notice that the only way the quick scan algorithm can take a bad step is if it steps into the middle of a low copy number repeat. The fact that the footprint is in the middle of a repeat could then go undetected, and a subsequent step out of that footprint might land outside the markers being connected. The inter-marker assembly algorithm gets around that problem by expanding a footprint as much as possible before any steps are taken from it. Thus if a footprint is within a low copy number repeat, it will be detected when the expansion reaches the edge of the repeat. And so the inter-marker assembly algorithm will advance safely across the genome.

8.1 Repetitive Footprints

The inter-marker assembly algorithm must be robust enough to backtrack through low copy number repeats. Recall that the quick scan algorithm will fail to find a connecting walk if the sequence between the markers contains a repeat that cannot be

stepped over. The one exception to this is if the repeat is unknowingly stepped into and a lucky step out comes from the correct occurrence of the repeat. We would like the inter-marker assembly algorithm to be able to get across such large, low copy number repeats. We accomplish this by augmenting the data structure for footprints to provide for what we shall call repetitive footprints.

A repetitive footprint is one that lies completely within a low copy number repeat and it is marked *REPETITIVE* in the inter-marker assembly algorithm. A repetitive footprint represents a low copy number repeat and it consists of all reads that lie within any occurrence of that repeat. Since the repeat will occur several times along the source, the repetitive footprint does not have a relative location as other footprints do. The relative positions of the reads contained within the footprint are known, however, as is the length of the footprint. Additionally, repetitive footprints contain a list of left and right *branches*. These branches correspond to the portion of the source on either side of each occurrence of the repeat (see Figure 7-1). How these branches are represented and used is detailed below.

Suppose a low copy number repeat has been stepped into. Expansion of the footprint will eventually reveal this fact by either detection of the interior edge of the repeat or by a database query that returns an inconsistent set of reads. Note that if the database query returns an inconsistent set of reads the repeat edge can still be determined through examination of these reads. When an expansion determines that a footprint lies within a low copy number repeat, the footprint is marked *REPETITIVE* and the database is queried again, this time using the repeat edge as an end point of the queried sequence.

This database query will return an inconsistent set of reads. All reads that don't cross the repeat edge (i.e. lie completely within the repeat) are added to the footprint. The rest of the reads are partitioned into maximal consistent subsets. These subsets are the *branches* of the repetitive footprint. If the repeat edge is on the left side of the footprint we call these the left branches; otherwise they are called the right branches. In general the number of these branches will be equal to the number of occurrences of the repeat, but occasionally an unfortunate coverage gap or pair of tandem repeats may cause the number of branches to be smaller than the number of occurrences of the repeat. From each of these subsets the read that extends furthest beyond the repeat is saved as a representation of that branch.

Suppose that a footprint f within a walk abuts (i.e. is a branch of) a repetitive footprint. We would like to know which branch on the opposite side of the repetitive

footprint corresponds to the same occurrence of the repeat as f . If a footprint g lies along that corresponding branch, then the repetitive footprint acts as a bridging, linking f to g . Left and right branches that come from the same occurrence of a repeat are said to be *correlated*, and these correlations, when discovered, are recorded within the *REPETITIVE* footprint. We can use the relative locations of footprints lying on opposite sides of a repetitive footprint r , together with the length of the r to determine if the footprints represent correlated branches of r . Note that branches could be correlated erroneously if two copies of the same repeat lie within a region smaller than the uncertainty of the locations of the footprints, but we expect this to happen very rarely.

In the inter-marker assembly algorithm, whenever an expansion reveals that a footprint lies within a low copy number repeat, the footprint is marked as *REPETITIVE* and the branches are set up as previously described. If an expansion reveals that a footprint abuts a low copy number repeat, then the algorithm checks to see if a repetitive footprint corresponding to that repeat exists. If it does not, it is created as described above. If it does exist, a search is conducted for a correlated branch. Each read representing an uncorrelated branch on the opposite side of the footprint is checked to see if it lies in a footprint between the current markers. If it does, the relative locations of the footprints are used to see if they are correlated.

To enable the creation of repetitive footprints, we introduce a new control value called *TOO_MANY*. This value is used to differentiate between ubiquitous and low copy number repeats. For example, the inter-marker assembly algorithm's version of the *take step* operation rejects a step only if more than *TOO_MANY* overlaps are found (indicating

one is in the middle of an ubiquitous repeat). The reason for defining this number is to allow for a buffer between the number of overlaps at which one calls a repeat ubiquitous, and the maximum number of overlaps returned by a database query, *MAX_OVERLAPS*. That is, due to statistical variation in the coverage there may be portions of a repeat that return just under *TOO_MANY* overlaps and others that return over *TOO_MANY*. By choosing enough of a gap between the values *MAX_OVERLAPS* and *TOO_MANY* one can effectively guarantee that once a repetitive footprint is created, subsequent expansions will include all its reads. For our tests we set these values to $20 \times \text{COVERAGE}$ and $10 \times \text{COVERAGE}$ respectively.

8.2 The First Pass

In the first pass of the inter-marker assembly algorithm maximal footprints are created between the markers. The first step after a new footprint is created is to expand it as far as possible in both directions, thus revealing whether or not the footprint is repetitive. In this pass only unique (non-repetitive) footprints are used to create new footprints. Repetitive footprints are created and correlated branches are searched for as outlined in the previous section, but no steps are taken from a repetitive footprint. The algorithm is outlined below:

Algorithm Inter_Marker Assembly (*lmark*, *rmark*: markers)

- Query the database with *lmark* to create a footprint *L*
- Expand *L* to the right as much as possible. The expansion ends because:
 1. A repeat that cannot be spanned by a read is encountered.

2. A gap in the coverage is encountered.
 3. The target marker is reached.
 4. The distance covered is more than $DIST$, an upper bound on the distance between the markers.
- **If case 3 or 4 above is true then**
 - Report either (3.) success and give assembly, or (4.) failure, and stop
 - **Else if the exterior edge of a low copy repeat edge is detected (i.e. there are not “too many overlaps”) then**
 - Create the repetitive footprint that corresponds to the repeat.
 - Expand L to the left until a gap or repeat is reached, or until L is too large to be stepped over. (i.e. L is longer than $LONG_LEN \times (1 + LEN_VARIATION)$)
 - Create a footprint R containing the right marker $rmark$ and repeat the process immediately above for L except in the opposite direction.
 - Place L and R as the initial elements in a FIFO *Queue*
 - *Pop_and_Walk*

Where *Pop_and_Walk* is a procedure defined as follows:

- **While *Queue* not empty do**
 - Pop a footprint f from the *Queue*.
 - **While** there is a step from f to a point whose estimated relative location is not less than 0 or greater than $DIST$ **do**
 - Take the step and expand the new footprint, g to the left and right while possible.

- **If** the expansion reveals a low copy number repeat and there is no repetitive footprint corresponding to it **then** create the repetitive footprint
- **Else if** the expansion bumps into a repetitive footprint, check for a correlated branch on the other side of the repetitive footprint
- **If** at any time L and R are connected, $DIST$ is set to be the estimated distance between the markers based on the connecting walk and recalculate the locations of all the unique footprints relative to $lmark$.
- **If** g is not *REPETITIVE* **then**
 - * Add g to the *Queue*.

One advantage to this algorithm compared to the quick scan is that in the algorithm above footprints never have to be merged. This is because all footprints are expanded as far as possible as soon as they are created. Thus every unique footprint ends with a gap or a repeat edge. The one possible exception to this is the left end of footprint L and the right end of footprint R . This is one reason why we expand these two footprints the opposite direction of the target marker. We wish to avoid dealing with the special case where L or R is unknowingly stepped over and the expansion bumps into them. The other reason to expand away from the target marker is to increase the chances of finding a confirmed step that lies close to the footprint.

When the algorithm above completes one has a linked set of footprints. This will be a connected set if and only if a walk joining the markers has been found. Notice that

such a walk will fail to be found if there are two low copy number repeats which are too long to step over lying between the markers. In this case the walk from each direction ends in a repetitive footprint that has a unique footprint as one branch (see Figure 8-2). This branch has no correlated branch. Searching for the correlated branches for each unique footprint that abuts a low copy number repeat is the next phase of the inter-marker assembly algorithm.

8.3 The Backtracking Search

Define as *unbridged* all unique footprints between the markers that are a branch of some repetitive footprint and have no correlated branch. To find the correlated branches for these unbridged footprints, a recursive routine called *branch_all* is defined. This routine searches all possible branches for the one that corresponds to the same occurrence of a repeat. The function is recursive since several levels of branch searching might be need to be conducted. For example, suppose there are three low copy number repeats between the markers, all of which are too large to be stepped over. In this case

searching the branches of the middle repeat will be required to reveal the correct branch of a search of the outer repeats (see Figure 8-3). To keep track of the depth of the search a level is passed to the *bridge_all* algorithm and all footprints created by it are marked with that level. Thus all unique footprints have an integer value called *LEVEL* attached to them. The value of *LEVEL* for all footprints created in the first phase is set to 0. Then the *bridge_all* routine detailed below is called with *level* = 1. For example, in Figure 8-3, the correct branch of the rightmost repeat is being searched. The footprints created during this search are given a level number of 1. Note that a level 2 search of the branches of the central repeat will be required in order to connect the walks.

function *bridge_all* (*level* : integer) : integer

- **Repeat**

- Find a unique footprint *F* with *LEVEL* = *level*-1, which is an uncorrelated, untried branch of some repetitive footprint *R*. If more than one such footprint exists, choose the one with the fewest possible correlating branches.
- If no such *F* exists, then return *EXHAUSTED*

- **FOUND := FALSE**
- **For** each read r that corresponds to an uncorrelated branch on the opposite side of R from F **do**
 - Explore the walk W corresponding to r
 - **If** W links to a footprint G with $LEVEL < level$ **then**
 - * **If** the locations of W and G are consistent **then**
 - **FOUND := TRUE**
 - $link := LEVEL$ of G
 - **Break** (out of for loop)
 - * **Else**
 - Destroy W
 - **Continue** (back to top of for loop)
 - $link := bridge_all(level+1)$
 - **If** $link \neq EXHAUSTED$ **then**
 - * **FOUND := TRUE**
 - * **Break** (out of for loop)
 - Destroy W
- **End For Loop**
- **If** **FOUND = TRUE** **then**
 - Record correlation between F and W in R
 - Set the $LEVEL$ of all unique footprints in W to $level-1$

- **If $link < level-1$ then return $link$**
- **If walks of opposite directions have been linked then**
 - * reset all locations to be relative to $lmark$
 - * set $DIST$ to estimated distance between markers
- Place all unique footprints in W on the *Queue*
- *Pop_and_Walk*
- **Else mark appropriate side of F as *FAILED***

Since the number of footprints involved in a walk is relatively small, a simple sequential search is used to find the footprint F in the first step of *bridge_all*. To facilitate this search each unique footprint records the status of its edges. That is we record whether the footprint ends in a gap, ubiquitous repeat, or low copy number repeat. In the case of a low copy number repeat, a link to the corresponding repetitive footprint is saved together with the correlating branch if known. This is also where the *bridge_all* routine would record *FAILED* if a correlating branch is not found. By ‘untried’ we simply mean the value *FAILED* does not appear here (i.e. the correlation has not already been searched for). We choose the F with the fewest possible correlating branches to minimize the number of branches we must search. For example in Figure 7-2 the branches of the repeat on the right would be searched first.

To execute the line ‘Explore the walk W corresponding to r ’, we first create a footprint B containing r . This is done by doing a database query on the portion of r that lies outside the repeat corresponding to R . B is then expanded as much as possible and then added to the *Queue*. A modified version *Pop_and_Walk* is then executed. In this

version a check is run after each footprint is created to see if it links to existing footprint whose level is smaller than the current branch level. This link can be in the form of a direct confirmed link, a correlated branch, or a link through a repetitive footprint other than *R*. If such a link is found, then the *Queue* is flushed and the checks for location consistency are performed.

The other modification to *Pop_and_Walk* further restricts which new footprints are created. In the original version of *Pop_and_Walk* a footprint is created if it will have a location between 0 and *DIST*. If this much freedom is given here, expanding an incorrect branch can result in many levels of branches needing to be searched before all possibilities are exhausted. This is because branches that lie outside the markers have no existing footprints in the area to bump into and limit the size of the walk. In general it is not very useful to search footprints that lie on the same side of *B* as *R*. So we will make the edge of *B* that abuts *R* one boundary for the creation of new footprints. The other boundary will depend on whether the walks have been joined or not. If they have not been joined, the other boundary is the extreme edge of the walk (i.e. 0 or *DIST* depending on which side of the repetitive footprint *B* lies on). If the walks have been joined, then the boundary is location of the edge of the unique footprint that lies closest to, but beyond the outer edge of *B*. By the walk *W*, we simply mean the linked set of footprints created by this process. For example, in Figure 7-4 the walks were joined during the first pass. In the routine *bridge_all* the left branches of footprint *B* will be searched for the branch that correlates to footprint *C*. The boundaries for the footprints built during this

exploration will be the location of the right edge of footprint *A* and the left edge of footprint *B*.

Notice that if a walk *W* fails to link validly with the some footprint that has a lower level than *W* (i.e. the branch does not correlate with the footprint *F*), then *W* is ‘destroyed’. Conceptually this means that the footprints in *W* are broken up and the reads that compose them are freed. In reality, this would be a terrible waste of work as these footprints are likely to be built again. So we add a status field to the structure of the footprint that is set to *DEAD* if the footprint is not part of any walk. We need to augment the *take step* routine to look for confirmed links to footprints that are marked *DEAD*. These footprints are stepped to simply by giving them a new relative location based on the footprint being stepped from, and changing the status field to *ACTIVE*. This will save many database queries.

The *bridge_all* routine attempts to correlate all branches at the level *level-1*. Notice that if part of the walk *W* links to a unique footprint with a level *L*, which is lower than *level*, then we check to see if the locations of the linked footprints are consistent. If they are not, we know that *W* is exploring an incorrect branch and we move on to explore the next branch. If the link is consistent, then *W* and all walks connected to it can assume the level *L*. This causes the unwinding of the recursion until we reach the instance of

bridge_all that was called with $level = L+1$. For example, in Figure 7-3, *bridge_all* first searches for the branch correlated to footprint *C*. This results in a second call to *bridge_all* with $level = 2$. This will search for the branch correlating to footprint *B*. Eventually (hopefully) the correct branch will be explored and a footprint that abuts the leftmost repetitive footprint will be created. This will cause a link to footprint *A*, which has $level = 0$. Now the levels of both walks will be set to 0, and all branches between the markers will have been successfully correlated. It is possible that a correlated branch will not be found. The most common cause of this is if the low copy number repeat lies next to a ubiquitous repeat. This prevents the initial footprint *B* of the correct branch from being created.

Upon completion of the *inter-marker assembly algorithm* we have a linked set of maximal footprints. If the *inter-marker assembly algorithm* successfully finds a walk linking the markers, the linked set will be connected. The links between the footprints form only a partial order on the footprints. One can obtain a complete order by using the estimated locations of the footprints. In general this ordering will be correct, though due to the variation in the length of long inserts it is possible for the position of two adjacent footprints to be transposed if their size is sufficiently small. Such a situation occurred in .15% of the test cases. In these tests two adjacent footprints, each less than 1k bases in length, had their positions flipped.

9 EMPIRICAL RESULTS

All empirical results were obtained using the simulator described in chapter 6.

The default values for the variables are:

<i>GENOME_LEN</i>	100M bases
<i>NUM_CLASSES</i>	2
<i>REPEAT_LEN[0]</i>	300 bases
<i>REPEAT_PERCENTAGE[0]</i>	20%
<i>REPEAT_LEN[1]</i>	1000 bases
<i>REPEAT_PERCENTAGE[1]</i>	5%
<i>MARKER_LEN</i>	300 bases
<i>MARKER_SPACING</i>	100K bases
<i>MARKER_VAR</i>	35%
<i>LONG_LEN</i>	1000 bases
<i>LEN_VARIATION</i>	15%
<i>READ_LEN</i>	400 bases
<i>COVERAGE</i>	10
<i>MIN_OVERLAP</i>	35 bases
<i>LS_RATIO</i>	1.0
<i>LONG_ERR</i>	5%

These values are used in the tests unless stated otherwise. The repeat classes here model small interspersed nucleic elements (SINEs) and long interspersed nucleic elements (LINEs). In the default settings there is no clustering of repeats. The default value for the length of short fragments is given by the following formula:

$$SHORT_LEN = \frac{2 \times READ_LEN - MIN_OVERLAP}{1 + LEN_VARIATION}$$

Tables 3 through 5 explore how changing the values of various model variables effects the success of the algorithm. All these tables have the same form. The leftmost column is the values of the model variable being examined. The next nine columns give the success rate of the algorithm based on various values for the maximum number of database queries allowed. These values range from 100 to 500 queries. The value given

is the percentage of times the algorithm finds a connecting walk between adjacent markers within the allowed number of database queries. The last column is the average number of database queries used by the algorithm. This value will actually vary according to the maximum number of database queries allowed, but the variance is usually quite slight. This is because in most cases the algorithm finds a connecting walk before the maximum is reached, so raising the maximum has little effect on the average number of queries. The average listed is calculated using a maximum of 250 database queries.

The first table, Table 3, examines the effects of different coverages. As would be expected, decreasing the coverage decreases the success rate of the algorithm, with the largest drop off at coverages of 4 and 5. It is encouraging to note that a success rate of 99.9% is possible with a coverage of 10. The average number of database queries increases as the coverage decreases. This is also expected since the algorithm is more likely to run into dead ends (footprints with no possible steps) with a lower coverage. The one exception to this is that the average database queries decreases as coverage goes from 4 to 3. This is because walks run out of total options to explore quickly at such a low coverage.

	Maximum Database Queries									
Cov	100	150	200	250	300	350	400	450	500	ave dbq max 250
10	94.5%	99.3%	99.7%	99.8%	99.8%	99.9%	99.9%	99.9%	99.9%	55
9	92.0%	98.9%	99.5%	99.6%	99.6%	99.6%	99.6%	99.6%	99.6%	58
8	88.2%	98.4%	99.1%	99.4%	99.4%	99.5%	99.5%	99.5%	99.6%	64
7	81.0%	96.2%	98.3%	98.5%	98.7%	98.7%	98.7%	98.7%	98.7%	73
6	70.9%	91.8%	95.9%	96.8%	97.0%	97.2%	97.2%	97.3%	97.5%	86
5	51.7%	76.5%	86.3%	88.7%	89.3%	90.0%	90.4%	91.4%	92.1%	114
4	17.4%	33.8%	44.6%	51.9%	57.2%	61.8%	65.1%	66.5%	66.6%	168
3	0.7%	3.0%	5.4%	7.9%	8.8%	9.1%	9.1%	9.1%	9.1%	96

Table 3: Quick Scan success rate with varying coverage

Table 4 examines the effects of differing marker spacing distances. As would be expected, increasing the marker spacing increases the number of database queries.

	Maximum Database Queries									
Dist	100	150	200	250	300	350	400	450	500	ave dbq max 250
50k	98.8%	99.8%	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	37
100k	94.5%	99.3%	99.7%	99.8%	99.8%	99.9%	99.9%	99.9%	99.9%	55
150k	81.1%	98.2%	99.3%	99.5%	99.6%	99.7%	99.8%	99.8%	99.8%	72
200k	62.9%	93.9%	98.5%	99.2%	99.4%	99.5%	99.7%	99.8%	99.8%	90

Table 4: Quick Scan success rate for different distances between markers

Table 5 examines the effects of the ratio of long to short inserts. More long inserts mean more possible steps. More short fragments mean longer reads, and thus wider footprints. The base case is 1:1 (equal proportion of longs and shorts). Increasing the proportion of shorts slightly decreases the effectiveness of the algorithm. Increasing

the proportions of long fragments (above 1:1) seems to have no real effect on the algorithm when the maximum database queries is at least 200.

	Maximum Database Queries									
long:short	100	150	200	250	300	350	400	450	500	ave dbq max 250
1:4	80.1%	97.2%	98.4%	98.9%	99.1%	99.1%	99.1%	99.2%	99.2%	78
1:2	91.9%	98.6%	99.2%	99.5%	99.6%	99.6%	99.6%	99.6%	99.6%	63
1:1	94.5%	99.3%	99.7%	99.8%	99.8%	99.9%	99.9%	99.9%	99.9%	55
2:1	94.0%	99.0%	99.7%	99.8%	99.8%	99.9%	99.9%	99.9%	99.9%	52
4:1	90.9%	98.5%	99.7%	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	53

Table 5: Quick Scan success rate with varying ratio of long to short fragments

Table 6 and Table 7 examine the effect of repeat clustering on the quick scan algorithm. One should observe first that even in the case that $CLUSTER_PROB[i] = .0$, clustering of the repeats does occur simply by the stochastic nature of placement under uniform probability. In the tables we consider increasing the degree of clustering for the

	Maximum Database Queries									
Prob.	100	150	200	250	300	350	400	450	500	ave dbq max 250
.1	94.8%	98.9%	99.6%	99.7%	99.7%	99.8%	99.8%	99.8%	99.8%	55
.2	94.0%	98.8%	99.4%	99.5%	99.6%	99.6%	99.6%	99.6%	99.7%	56
.3	93.1%	98.9%	99.4%	99.6%	99.7%	99.7%	99.7%	99.7%	99.7%	56
.4	91.8%	98.5%	99.2%	99.5%	99.5%	99.5%	99.5%	99.6%	99.6%	58
.5	90.9%	98.2%	98.9%	99.3%	99.4%	99.4%	99.4%	99.5%	99.5%	59

Table 6: Quick Scan success rate with varying clustering, cluster density is 1.5

default data set by varying the cluster probability, *CLUSTER_PROB*[*i*] from .1 to .5 in increments of .1. In Table 6 we set the density, *CLUSTER_DENSITY*[*i*], to 1.5. In Table 7 this parameter is set to 2.0. The parameters are set similarly for both repeat types and when *CLUSTER_PROB* = .5, repeats occur in clusters of average size 2. Observing the table reveals that clustering does not substantially affect the results.

	Maximum Database Queries									
Prob.	100	150	200	250	300	350	400	450	500	ave dbq max 250
.1	94.7%	99.1%	99.6%	99.6%	99.7%	99.8%	99.8%	99.8%	99.8%	55
.2	94.7%	99.0%	99.5%	99.7%	99.8%	99.8%	99.8%	99.8%	99.8%	55
.3	94.7%	99.0%	99.5%	99.7%	99.7%	99.8%	99.8%	99.8%	99.8%	55
.4	93.6%	98.8%	99.4%	99.5%	99.6%	99.7%	99.7%	99.7%	99.7%	56
.5	94.0%	98.9%	99.4%	99.6%	99.7%	99.7%	99.7%	99.7%	99.7%	56

Table 7: Quick Scan success rate with varying clustering, cluster density is 2.0

Table 8 examines the effects of longer repeats. Each row gives the success rate for the algorithm when a repeat occurrence of the given length appears between the target markers. This repeat occurrence is in addition to the 5% LINEs and 20% SINEs set up by the default parameters. Lengths of less than 5K had practically no effect on the success rate and thus are not shown. Since the average length of long fragments is 10K, repeats of length 10K or higher cannot be stepped over.

For the inter-marker algorithm, that has the capability of resolving low copy number repeats, we augmented the default model to involve a somewhat more complex

	Maximum Database Queries									
Length	100	150	200	250	300	350	400	450	500	ave dbq max 250
5k	84.0%	97.7%	99.0%	99.2%	99.3%	99.3%	99.4%	99.4%	99.4%	69
6k	77.7%	96.5%	98.8%	99.1%	99.2%	99.2%	99.2%	99.2%	99.3%	75
7k	70.1%	93.3%	97.9%	98.3%	98.5%	98.5%	98.5%	98.5%	98.5%	83
8k	64.6%	91.6%	97.0%	97.7%	97.8%	97.8%	97.9%	97.9%	97.9%	89
9k	54.1%	83.1%	90.6%	91.5%	91.9%	92.0%	92.1%	92.2%	92.2%	106

Table 8: Quick Scan success rate over intervals that contained a long repeat

repeat model parameterized by three variables X , M , and N . In addition to the ubiquitous SINE and LINE repeats, $X\%$ of the genome is further constructed to consist of low copy number repeats, each occurring in copy number uniformly chosen to be between 2 and 5. The length of each class of low copy number repeat is uniformly chosen to be between N and M thousand bases long. The tables 9 through 13 consider each choice of X in $\{2.5\%, 5.0\%, 7.5\%, 10.0\%\}$ and each choice of $[N, M]$ in $\{[1, 5], [5, 9], [10, 20], [20, 40]\}$, with the exception of the quick scan algorithm that always fails when $N \geq 10$.

Table 9 gives the success rate for the quick scan algorithm under these conditions. In producing these values, we allowed the algorithm a maximum of 500 data base queries. These results are given to illustrate the relative strength imbued by the backtracking capability given to the inter-marker algorithm. As would be expected the success rate drops slightly when the size of the repeats gets longer than what can be spanned by the long inserts.

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	99.4%	99.2%	99.1%	99.1%
5-9k	99.4%	98.6%	97.6%	94.7%

Table 9: Quick Scan success rate using more complex repeat structure

Table 10 gives the scaffold success rate for the inter-marker assembly algorithm with these repeat models. That is, the percentage of adjacent markers that can be walked between. Notice that as expected this rate is higher than the success rate for the quick scan algorithm. Some of these success rates are as high as 100%. This is not to say that the algorithm cannot fail in those situations, but rather in our test cases success occurred in at least 99.95% of the time. Due to the computational intensity of the tests it was not possible to run enough cases to reasonably report the results to a higher precision than a tenth of a percent.

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	100.0%	100.0%	100.0%	100.0%
5-9k	100.0%	99.9%	99.9%	99.9%
10-20k	99.9%	99.9%	99.7%	99.5%
20-40k	100.0%	100.0%	99.9%	99.8%

Table 10: Success rate of inter-marker assembly algorithm

One part of Table 10 that might be surprising is that the success rate increases as the size of the low copy number repeats increases from 10-20k to 20-40k. With a little thought though, this would be expected. Since the average length of long inserts is 10k, when the repeat size reaches 10k it can no longer be stepped over. So one would expect the success rate to drop when repeats of at least this size are included. To deal with these larger repeats the algorithm must rely on searching branches. The two most common sources of failure for these walks is an archipelago of unique footprints that are too small to have confirmed links, and a branch of a unique footprint that cannot be expanded because the repeat occurrence is adjacent to an ubiquitous repeat. Notice that neither of these cases are affected by the size of the repetitive footprints, but both are affected by the number of repeats. Since it requires twice as many repeats of size 10-20k than repeats of size 20-40k to cover the same amount of the genome, one should expect the success rate to be higher for the larger repeats.

The next three tables give statistics for those cases where the inter-marker assembly algorithm found a successful walk. The first, Table 11, gives the percentage of the unique sequence between the markers covered by the inter-marker result. Notice that this number remains uniformly high for all the repeat schemes tried.

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	100.0%	99.9%	99.9%	99.9%
5-9k	100.0%	99.9%	99.9%	99.8%
10-20k	100.0%	99.9%	99.9%	99.8%
20-40k	100.0%	99.9%	99.9%	99.9%

Table 11: Percent of unique sequence covered by inter-marker assembly algorithm

Table 12 gives the percentage of the sequence between the markers characterized by the inter-marker result. A sequence is characterized if it appears in any footprint, repetitive or unique, placed by the inter-marker assembler. If the repeats were truly identical, than this would be the percentage of the source sequence assembled. Once again the numbers remain high independent of the repeat model used.

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	100.0%	99.9%	99.9%	99.9%
5-9k	100.0%	99.9%	99.9%	99.8%
10-20k	100.0%	99.9%	99.9%	99.8%
20-40k	100.0%	99.9%	99.9%	99.8%

Table 12: Percent of sequence characterized

Lastly, Table 13 gives the percentage of the sequence between the markers uniquely assembled by the inter-marker result. Any part of the source that is covered by a unique footprint is considered uniquely assembled. In addition, for every placement of a repetitive footprint, we search for reads in that footprint which have a mate pair confirming the read comes from that particular repeat occurrence. The sequence covered by such reads is also considered to be uniquely assembled. Notice that the percent uniquely assembled is guaranteed to be smaller than the percent characterized. Also notice how the number drops as the size of the repeats gets large. These numbers are exactly what we would expect since the average size of a large insert is only 10k bases long, one third of a repeat of length 30k cannot possibly be uniquely assembled.

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	99.9%	99.9%	99.8%	99.7%
5-9k	99.9%	99.8%	99.7%	99.5%
10-20k	99.7%	99.4%	99.1%	98.7%
20-40k	99.0%	97.9%	96.9%	95.7%

Table 13: Percent of source uniquely sequenced

10 THE CELERA GENOME SEQUENCING PROJECT

As described in section 4.6, Celera Genomics plans to use the whole genome shotgun sequencing method to sequence the entire human genome by 2001. This author had the privilege to work for several months with the group at Celera that is building the program to assemble the genome. This allowed access to data and code that could not have otherwise created for this thesis due to time and resource constraints. The assembler being built at Celera does not use the algorithms in this thesis, although some of the ideas presented here are incorporated into it. The Celera assembler is the work of many scientists, and will probably be the source of several scholarly articles. It will not be described in detail here. However, we will outline some of the differences between the Celera approach and that described earlier in this thesis. We will also give a description of some of the methods and data structures used, so that we can see how they can be used to further test the algorithms given in this thesis. A description of the completed Celera assembler has been very recently published.³⁵

10.1 The Shotgun Step

Unlike the whole genome shotgun method proposed in this thesis, Celera does not intend to size-select inserts that are small enough to have the end reads overlap. It was decided that the benefit of having a longer read (the effective result of inserts of type *SMALL*), was not as great as the information given by having mate pairs. So Celera will end sequence two classes of inserts that are both long enough to give mate pairs. One class will be size-selected to be around 2K bases long and the other to be around 10K

bases long. This does not mean that all reads will have a mate pair. In the lab, it sometimes happens that a sequencing reaction fails. Celera expects to successfully sequence both ends on an insert about 70% of the time. This means that around 30% of the reads will not have a mate. Originally, it was thought that the greater expense of working with 10K inserts would require a 4:1 ratio or 2K to 10K inserts. However, lab experience has since indicated that the added expense of 10K inserts was not as great as anticipated and a ratio of 1:1 is being used. The experiments for this thesis were done when a 4:1 ratio was still planned.

We modified the simulator described in section 5 to test the inter-marker assembly algorithm on this new model of inserts. This amounted to adding a second insert of type *LONG*. Thus, the variable *LONG_LEN* became *LONG_LEN1* and *LONG_LEN2*. The variable *LS_RATIO* was replaced by variables *PERCENT_SHORT* and *PERCENT_LONG1* that give the percentage of the reads that were of type *SHORT* and *LONG1* respectively. Obviously, the percentage of reads of type *LONG2* is one minus those percentages. The value for *SHORT_LEN* is set to *READ_LEN*. The results of these tests are given in the following tables. These tables are analogous to Table 10 through Table 13. The values for some of the parameters used for these tests is given below:

<i>READ_LEN</i>	550 bases
<i>LONG_LEN1</i>	10,000 bases
<i>LONG_LEN2</i>	2,000 bases
<i>SHORT_LEN</i>	<i>READ_LEN</i>
<i>PERCENT_LONG1</i>	14%
<i>PERCENT_SHORT</i>	30%
<i>LONG_ERR</i>	1%

These values imply the percentage of 2K inserts is 56%. These numbers reflect a ratio of 2K to 10K inserts of 4:1 with a 30% failure rate for obtaining both mate pairs. Notice we used a larger value for the read length than in our other tests. This value is a

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	99.9%	99.9%	99.8%	99.7%
5-9k	100.0%	99.9%	99.7%	99.6%
10-20k	99.9%	99.9%	99.9%	99.5%
20-40k	100.0%	100.0%	100.0%	99.9%

Table 14: Success Rate Using of Inter-marker Assembly Using Celera's Insert Model

more optimistic and hopefully realistic value for the Celera labs than the conservative value of 400 bases used as the default in earlier tests. In addition, because Celera is using capillary gel machines, the rate of mismatched mates is dropped to 1%. The values for

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	100.0%	100.0%	100.0%	99.9%
5-9k	100.0%	100.0%	100.0%	100.0%
10-20k	100.0%	100.0%	100.0%	99.9%
20-40k	100.0%	100.0%	100.0%	100.0%

Table 15: Percent of unique sequence covered by inter-marker assembly algorithm using the Celera model

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	100.0%	100.0%	100.0%	99.9%
5-9k	100.0%	100.0%	100.0%	100.0%
10-20k	100.0%	100.0%	100.0%	99.9%
20-40k	100.0%	100.0%	100.0%	100.0%

Table 16: Percent of sequence characterized using Celera Model

the parameters not listed above are the same as given in the list of default values in Section 8.

Notice that the success rate is still very high for this model. The biggest difference between the tables above and Table 10 through Table 13 is the somewhat lower values for the percentage of the source uniquely sequenced given in Table 17. This is expected since for this model only 14% of the inserts are 10K as opposed to 50% in the

Repeat	% of Genome Covered			
Size	2.5%	5.0%	7.5%	10.0%
1-5k	99.9%	99.8%	99.8%	99.6%
5-9k	99.7%	99.4%	99.1%	98.6%
10-20k	99.3%	98.6%	97.8%	97.0%
20-40k	98.5%	97.1%	95.6%	94.0%

Table 17: Percent of source uniquely sequenced using Celera model

previous model. Thus, we expect to have less success in covering the interior of large repeats with an anchored mate pair.

10.2 The Celera Assembler and Unitigs

The Celera assembler begins with an overlap phase as described in Section 4.3. When designing the algorithms in this thesis, however, we assumed the overlap graph was too expensive to build. There are two significant reasons for this for these differing approaches. The first is simply that Celera has more computing power than we had anticipated being available for this project. The second reason comes from a fundamental difference in the overall sequencing schemes used and their relation to getting intermediate results out quickly. Our method assumes all the sequencing is already completed when the assembler is invoked. Thus, the intermediate results are the assemblies between markers. Rather than take the initial block of time to build an overlap graph (assuming it was possible) for the entire genome, we are able to get results out faster by just concentrating on the comparisons needed for assembling the genome

between the markers. Celera, on the other hand, desires its assembler to be running in parallel with the sequencing effort. Intermediate results are entire assemblies of the fragments sequenced to date. Clearly, this assembly will improve as the coverage increases. Note that this means that the program doing the overlapping will be getting the reads in relatively small increments. Thus, while it may take weeks to build the overlap graph for all the reads needed for shotgun sequencing the entire human genome, it may only take a day to add to the overlap graph a batch (say a week's worth) of newly sequenced reads.

It should be clear that by not creating the overlap graph we can begin outputting the inter-marker assemblies more quickly and that we do not have to deal with a structure as huge as the overlap graph. What might not be immediately obvious is using the algorithms described in this thesis to assemble the reads requires fewer sequence comparisons than are required to build the overlap graph. By not building the overlap graph, we actually reduce the number of sequence comparisons by a factor equal to the coverage, C , minus one. To see this, remember that each expansion of a footprint causes a database query that returns an average of $C - 1$ reads. These reads are now placed in the footprint and, other than the read that extends furthest in the direction of expansion, never need to be compared against the rest of the database.

The second step of the Celera assembler is a simplification of the overlap graph and the subsequent creation of *unitigs*. This process was devised and described in detail by Myers in 1995,³⁶ where a unitig is called a *chunk*. A unitig is a maximal contig made

up of reads that have a unique relationship to each other in the overlap graph. An outline for the building of unitigs is as follows:

1. Remove from the graph any reads that are *contained* in another read. That is, remove reads for which there exists an edge in the overlap graph that places the read inside another.
2. Remove edges from the overlap graph that are transitively implied. To be precise, if there is a transitive relationship between three edges, then the one that indicates the smallest overlap between the reads is removed.
3. Form unitigs by joining reads that are uniquely connected by this revised graph. For example, if there is an edge in the overlap graph connecting the right side of read R to the left side of read S and there are no other edges from the right side of R or the left side of S , then R and S are joined together into the same unitig.
4. Replace each read removed in step 1 by adding a *containment* edge connecting the removed read to one of the reads containing it.

This outlines how to create unitigs, or chunks, as defined in the Myers' paper. The unitigs created by the Celera assembler are a bit more complicated. One problem is replacing the contained reads. There are cases where reads might be contained in reads located in different unitigs. This leads to ambiguity when replacing the contained reads in step 4. Placing a contained read in the wrong unitig causes extra problems for the Celera assembler because a read has a possible mate pair as well as sequence information. In addition, as will be described later, a status number is calculated for each

Figure 10-1: Normal unitigs fail to span a short repeat

unitig to determine its likelihood of coming from a unique portion of the genome. Having contained reads placed incorrectly would adversely effect the accuracy of this number. Therefore, the procedure Celera uses to build unitigs must incorporate contained reads in a more careful way than outlined in the Myers' paper.

A second shortcoming of unitigs as described is that they will not span repeats as the footprints do. For example, suppose there is an occurrence of a short repeat (like an ALU) that is spanned by some read, r_1 , in the database. This case is illustrated in Figure 10-1. In the overlap graph a read, s , whose right end is in this repeat will have an edge to r_1 , but also to all reads whose left end is in an overlapping section of any occurrence of the repeat. In Figure 10-1 r_1 overlaps r_2 and r_4 . The edge between r_1 and r_2 is removed during the transitive edge removal stage, but the edge between r_1 and r_4 remains. Thus

there is more than one edge from the right side of r_1 and so s and r_1 are not collapsed into the same unitig.

On the other hand, recall from Section 6.4 how footprints manage to span a repeat in this situation. Suppose a footprint including r_1 is being expanded to the right. If a database query is called using r_1 , then s , r_2 , and r_4 are all returned. These reads are inconsistent. However, the edge of the repeat is also reported. The database query is then called again using only the unique portion of r_1 . This query will return s and r_2 but not r_4 . Thus, the footprint has spanned the repeat.

When building unitigs as described in Myers' paper, for each occurrence of a short repeat there will be up to three unitigs. These include the unitig to the left of the repeat, a short unitig made up of all reads that span or are contained completely within the repeat, and the unitig on the right side of the repeat. The Celera uses a combination of examining the unitig graph together with repeat edge detection to combine these three unitigs. Therefore the unitigs created by the Celera assembler are generally longer than those described in the Myers' paper, and will usually span short repeats.

There is a value associated with every unitig called the *discriminator*. This number is used to help determine which unitigs cover unique portions of the genome. It is based on the coverage of the unitig and its length. The idea is that an unusually high coverage indicates a unitig is repetitive, and the longer the unitig, the more significant this evidence is. Owing to statistical variations in the coverage, it is impossible to differentiate between short unitigs that are unique and those that are repetitive. However, empirical evidence has shown that a boundary value exists such that all unitigs whose

discriminator exceeds that value are almost surely unique. Furthermore, all unique unitigs of significant length have a discriminator value that exceeds this boundary. Unitigs whose discriminator value exceeds this boundary are called *discriminator unique*.

The exact methods used to build these unitigs are both not this author's work and proprietary intellectual property of Celera Genomics, so they cannot be described in detail here. This much information has been given to demonstrate that the discriminator unique unitigs built by the Celera assembler are in theory, equivalent to the non-repetitive footprints built by the inter-marker assembly algorithm. They will be used as such to further test the usefulness and success rate of that algorithm as outlined in Section 10.4.

10.3 Celsim: Celera's Simulator

To test their assembler the Celera team built a simulator to generate source sequences and uniformly distributed inserts from the source. The program, called Celsim³⁷, uses a modified version of the frag program created by Gene Myers at the University of Arizona. It either uses a source sequence from a file, or creates its own based on input parameters. The source is uniformly broken up into subsequences (inserts). One can specify for these inserts a length, either by giving a range from which the lengths are uniformly chosen, or a mean and standard deviation can be given and the lengths are randomly selected using a normal distribution. A probability that the dual end sequencing will fail (and only the read from one end is produced) can be input, as well as a probability that the insert is chimeric. The length of the read is specified in the same manner as the length of the insert. Finally, the sequencing error rate of the reads can be

specified. The probability of an error for each end of the read is supplied, and errors are generated in a Poisson fashion based on a linear interpolation of the given error rates.

The Frag program uses a sophisticated stochastic grammar for generating a DNA sequence with the desired nucleotide composition and repeat structure. One starts by defining basis sequences. For these sequences one specifies the length, and the probability of occurrence of each nucleotide type. These basis sequences are then combined to form a more complex structure. For each instance of a basis in these sequences one can specify:

- **Orientation:** This is the probability that an occurrence of the specified sequence will be in reverse orientation and complimented.
- **Mutation:** This is a measure of the fidelity of each occurrence of the sequence. One gives a mutation rate, or a range of rates. If a range is given, then for each occurrence of the sequence a rate is uniformly chosen from the range. Each occurrence of the sequence is mutated by exactly this rate.
- **Repetition:** This allows one to specify how many occurrences of the sequences are included.
- **Fracture:** In real genomes, repeats are fractions of mutated copies of some original sequence. The fracture parameter allows one to specify what percentages of the occurrences are fractured, and the possible lengths and ranges of these fractures.

This grammar is extremely flexible and allows quite complex structures to be defined. Test files were created at Celera that were intended to closely reflect the

structure of the human genome. These files are quite large, though considerably smaller than the full human genome to allow programs to be tested in a reasonable amount of time. The data file used in our tests will be called is approximately 10^{-3} times the size of the human genome. The source sequence of this file therefore has a length of 3,500,000 bases. Approximately 10% of this sequence is covered with ALUs, which are 282 bases long, have a mutation rate of 5-30% and 12% of them are fractured with a fractured length of 20-95% of the original. Approximately 5% of the source is made up of LINES. These have a base length of 7,000 bases, however they are suffix fractured (the fracture consists of the last part of the sequence). 94% of these LINES have a fracture size of 5-20% of the original and 6% have a fracture size of 20-100% of the original. LINES have a mutation rate of 5-15%. Lastly, there is a large repeat representing two copies of a gene. This repeat has a size of 1000-5000 bases, a mutation rate of 1-5%, and is not fractured.

The markers used in these test files model *BAC ends*. These are the end sequences of BAC clones. The lengths of the BAC clones modeled are randomly chosen using a normal distribution. The average BAC length, *BAC_LEN*, was set to 150K bases and the standard deviation was set to 50K for these experiments. This gives a much larger variation in the distance between markers than was used in our earlier experiments. The probability BAC ends are mismatched was set to 20%. If a BAC clone is chimeric then the markers come from unrelated random parts of the source. There are enough BAC clones to give a 30-fold coverage of the source. Thus, the average distance between markers is 5K, though for each marker only the distance and relative orientation of its

mate pair on the BAC clone is known. In our previous model, the markers were assumed to come from a unique portion of the source, but in the current model the BAC clones are positioned along the source randomly using a uniform distribution. There is nothing to prevent a BAC end (marker) from appearing in the middle of a repeat. These markers are treated as reads by the module that builds the overlap graph and the one that builds the unitigs. Therefore, each marker is contained in some unitig.

10.4 Applying the Algorithms

We adapted the quick step and inter-marker assembly algorithms described in this thesis to use information from the Celera assembler. First, the Celsim program was used to generate the data file described in section 10.3. The Celera assembler was then invoked on this file to build the unitigs. These unitigs were treated as the footprints of the algorithms. The discriminator unique unitigs, described in section 10.2, were considered equivalent to the maximal unique footprints created by the inter-marker assembler as described in Section 8.

We start by building a *graph of unique unitigs*, $G = (N, E)$, where the nodes are the discriminator unique unitigs and the edges are the confirmed links. There is a confirmed link between two unitigs (footprints) if and only if the unitigs contain at least two sets of mate pairs with matching orientations. Let $G_1, G_2, \dots, G_n$ be the maximal connected subgraphs of G . Because we have an approximation for the distance between mate pairs, we can determine the approximate relative locations of the unitigs within a maximal connected subgraph. Therefore, for each unitig $c \in G$ we will define a value

location(c) to be the location of the left-most base of c in the context of G_i . Recall we know the relative location of every read r in c . This information together with **location**(c) allows us to calculate the location of r in the context of G_i . We will refer to this value as **location**(r).

Since the relative orientation of mate pairs is known, and the mate pairs define the links, the relative orientation of all unitigs within a maximal connected subgraph G_i is known. Therefore choosing an orientation for a single unitig in G_i will force an orientation on the rest of the unitigs in G_i . We will assume that for each G_i such an orientation is chosen. This choice will determine an orientation for every read $r \in c \in G$. We shall say that **orientation**(r) = *FORWARD* if r is positioned in c such that the 5' end of r is to the left of the 3' end, and **orientation**(r) = *BACKWARD* otherwise. The sequencing reaction always proceeds from the 5' end of the read to the 3' end. Therefore, if r_1 and r_2 are true mate pairs that lie within the same maximal connected subgraph, then **orientation**(r_1) $\neq$ **orientation**(r_2).

The goal of the quick scan algorithm presented earlier was to try to quickly confirm the adjacency of two markers. For the current model the goal becomes trying to determine whether or not a pair of BAC ends are bad. The algorithm will take as input a pair of mated BAC ends and a graph of unique unitigs, and will return a value giving the status of the BAC ends. The possible values of the status returned by the quick scan algorithm are *REPETITIVE*, *UNKNOWN*, *BAD*, and *CONFIRMED*.

The status is *REPETITIVE* if at least one of the BAC ends lies in a unitig that is not discriminator unique. This situation was not possible in our earlier model, where all

markers were assumed to come from unique portions of the genome. For this reason, coupled with the fact that the 30-fold coverage of BACs gives us plenty of markers, we will ignore BAC ends that are marked as *REPETITIVE* throughout the rest of our applications.

BAC ends are given a status of *UNKNOWN* if the quick scan algorithm cannot find a path connecting them. Recall that the quick step algorithm as defined in section 6 uses confirmed steps from footprints not known to be repetitive to try to traverse from one marker to the other. Notice that this algorithm will be successful here precisely when the BAC ends lie in the same maximal connected subgraph of G . Thus BAC ends are marked as *UNKNOWN* by the current version of quick step if and only if they do not lie in the same maximal subgraph.

Recall that when the quick step algorithm found a path connecting the markers, the estimated length of the path was compared with the given approximate distance between the markers to see if the path was accepted or rejected. Finding a path with a bad length could mean that the given approximate distance was incorrect, or that the path had taken a bad step by stepping into a repeat. In our current situation, we know that the footprints we are using are unique, and thus we assume the path is correct. We also must be aware of orientation here; a factor we did not simulate earlier. Therefore, if the BAC ends lie in the same maximal connected subgraph, then their status as *BAD* or *CONFIRMED* is determined by their distance and orientation.

Procedure Quick Step (G : graph of unique unitigs, m_1, m_2 : BAC ends) : status

- Find unitigs c_1 and c_2 such that $m_1 \in c_1$ and $m_2 \in c_2$
- If $c_1 \notin G$ or $c_2 \notin G$ (i.e. one of the unitigs is not discriminator unique) then
 - **Return *REPETITIVE***
- Find the maximal connected subgraphs G_1 and G_2 such that $c_1 \in G_1$ and $c_2 \in G_2$
- If $G_1 \neq G_2$ then
 - **Return *UNKNOWN***
- If $\text{orientation}(m_1) = \text{orientation}(m_2)$ then
 - **Return *BAD***
- If $|\text{location}(m_1) - \text{location}(m_2) - \text{BAC_LEN}| \leq \Delta$ then
 - **Return *CONFIRMED***
- **Else Return *BAD***

Recall that BAC_LEN is the average length of a BAC clone. The value of Δ is chosen so that most of the BAC lengths will fall into the range $\text{BAC_LEN} \pm \Delta$. For these test runs we let $\Delta = 100\text{K}$, which is 2 times the standard deviation of the BAC lengths. We therefore expect 98% of the BAC lengths to lie in the desired range.

We ran this test on our data file and obtained the following results:

Number of <i>REPETITIVE</i> BAC end pairs	113
Number of <i>UNKNOWN</i> BAC end pairs	132
Number of <i>CONFIRMED</i> BAC end pairs	90
Number of <i>BAD</i> BAC end pairs	11

In our original model for the genome outlined in section 4.1, we assumed that markers were comprised of unique sequence. That being the case, we will ignore *REPETITIVE* BAC end pairs in our current evaluation. Erroneous markers were not simulated in our earlier models either. In this case, however, 42 pairs of non-repetitive markers came from simulated chimeric BAC's. It should be noted that the above algorithm made no mistakes in labeling the markers with respect to their being erroneous. That is, no markers from chimeric BAC's were confirmed, and no good markers were labeled as *BAD*.

The quick scan algorithm was not originally intended to identify bad markers, but to quickly confirm a high percentage of good markers. The relevant value from the test above is therefore the percentage of good marker pairs confirmed. In the test there were 191 good marker pairs which lay within unique unitigs. Of these, 90, or 47% were confirmed. Based on our earlier tests, this number is surprising small. There are a number of reasons for this success rate being so much smaller than those reported in tables 3-9. First, the test data for this run included long repeats that could not be stepped over. Second, due to simulated sequencing errors, some overlaps were missing from the

overlap graph that would have been present in our earlier, more idealized simulations. The most significant reason for the drop in success rate is a limitation on the parts of the genome the algorithm described above is allowed to ‘step’ as compared to the algorithm described in section 6. The primary method of determining unique portions of the genome used in earlier chapters was by detecting the edge of repeats, thus short sections of the genome which lie between repeats can be stepped into. The discriminator used to distinguish unique unitigs generally does not allow the status of short unitigs to be determined.

The low success rate of this incarnation of the quick step algorithm should not be a large concern. Recall the purpose of the algorithm was to quickly confirm good markers so that the more time intensive inter-marker assembly algorithm could be run on them first. If these algorithms are invoked after the overlap graph and unitigs have been calculated, then the time intensive work has already been done. For our test data there is practically no time difference between running the quick step algorithm and the complete inter-marker assembly algorithm described below.

Recall that the goal of the inter-maker assembly algorithm outlined earlier is to produce an ordered list of maximal unique footprints connecting two adjacent markers. This list of footprints should cover as much of the source between the markers as possible. For the current model the goal is the same with mated BAC ends taking the place of the adjacent markers. We will only deal with BAC ends that are not marked *REPETITIVE* or *BAD* by the version of quick step given above. Since the discriminator

unique unitigs are equivalent to the maximal unique footprints, most of the work the inter-marker assembly has already been done.

Let G be the graph of unique unitigs and $G_1, G_2, \dots, G_n$ be the maximal connected subgraphs of G . If $n = 1$ then the location of each unitig in G can be used to completely order the set, and the goal of the inter-marker assembly algorithm is met for all marker pairs. So for the rest of this discussion we will assume that $n > 1$. Let m_1 and m_2 be mated BAC ends and c_i and c_j be discriminator unique unitigs such that $m_1 \in c_i \in G_h$ and $m_2 \in c_j$. The markers m_1 and m_2 are connected by a set of maximal unique footprints if and only if $c_j \in G_h$. This meets the goal of finding an ordered set of unique unitigs connecting the markers, but does not meet the goal of covering as much as the source between the markers as possible. As illustrated in Figure 10-2, a unique unitig may lie between unitigs in a maximal connected subgraph. Although the unitigs of G_h link the two markers, they do not cover as much of the unique portion of the source as possible. We would like some variation of the inter-marker assembly algorithm to connect these interleaved unitigs with the unitigs of G_h .

Figure 10-3: A repetitive footprint modeling a fractured repeat

The first pass of the inter-marker assembly algorithm described in Section 8.2 builds linked maximal unique footprints. This has already been done in our current context. The graph of uniques contains all the maximal unique footprints. The portion of the inter-marker assembly algorithm that has not yet been utilized is searching the branches of repetitive footprints. It is that search that might have detected a unique footprint not connected by confirmed links as illustrated in Figure 10-2. Unfortunately, the repetitive footprints we dealt with in Section 8 do not exist here. Instead, we must deal with what we shall call a *repeat blizzard*, which will be described next.

Under our earlier assumption that all occurrences of a repeat were identical, all reads contained in any occurrence of the repeat collapsed into a single repetitive footprint. This footprint, pictured in Figure 8-1, has branches on each end representing the unique portion of the source bounding each occurrence of the repeat. As described in Section 10.3, repeats simulated here might be fractured. This would cause the branches to be staggered across the repeat (see Figure 10-3).

Figure 10-4: A repeat blizzard

In addition to fracture, the repeats modeled here are mutated copies. This prevents the reads within the repeats from collapsing together into a single repetitive unitig covering the repeat. Instead, local differences cause reads from one occurrence to overlap with reads from some, but not all, other occurrences of the repeat. For example, suppose there were three occurrences of a repeat which we will label as A, B, and C. Then reads covering some portions of occurrence A will overlap reads from occurrence B, but not reads from occurrence C. Others will overlap reads from C, but not from B. Others will overlap reads from both and still others from neither. This causes ambiguity in how the fragments can be joined together. Therefore, the fragments covering the repeats fail to collapse into a single unitig.

The ultimate result of this overlap graph is what we call a *repeat blizzard*. A repeat blizzard is a large number of short unitigs (most contain 1-5 reads) with a tangled set of overlaps. Figure 10-4 illustrates a repeat blizzard. All the horizontal line segments represent unitigs. The unitigs marked 'unique' are discriminator unique. The others

probably cover some repeat in the source. Notice that the unique unitigs appear at slightly staggered distances because of repeat fracture. The lines connecting the unitigs are overlaps. This should not be confused with the links between unitigs in the graph of uniques, which come from confirmed mate pairs. This illustration is highly simplified. Real examples of repeat blizzards will usually have far more unitigs and overlaps than are pictured.

While the repeat is not covered by a single repetitive footprint with multiple branches, the structure is similar. In Figure 9-4 the three unique unitigs on the right side of the blizzard can be thought of as the right branches of this structure. These branches can be located by starting on a left branch (the unique unitig on the left) and searching the blizzard until a unique unitig is located. This allows us to carry on a search of the branches similar to that defined in Section 7.3. For example, suppose that there is a link in the graph of uniques between the unique unitigs on opposite ends of the repeat shown in Figure 9-4. Starting at one of these unitigs, we search for a path across the overlap graph for the other unitig. The links give an approximate distance between the unitigs. Since the length of the path searched is known, we have a bound on our search, and a method of verifying that the path found is viable.

The first step in our algorithm will be to search the repeat blizzards separating adjacent, connected unitigs. This will allow us to find some of the unique unitigs that lie between the connected unitigs, like the one illustrated in Figure 9-2. This search could be done for all unitigs connecting mated BAC ends, but we choose to take a more global view here and search between all connected unitigs. Thus, for each pair of adjacent

unitigs, c_i and c_{i+1} , in a maximal subgraph G_j , we do a bounded, depth first search over the repeat blizzard separating the unitigs. This will accomplish three things.

1. Most importantly, if the path connecting the unitigs contains a discriminator unique unitig $c_k \in G_k$, then c_k can be linked to c_i and c_{i+1} using the distances between them on the path. This allows G_j and G_k to be combined into a single connected subgraph.
2. Since the distance along the path is calculated using overlaps and sequence lengths, errors in this number will come from an imbalance of insertion and deletion sequencing errors. Thus, the distance calculated would be more precise than links, which are calculated from mate pairs. This gives a more accurate calculation for the distance between the unitigs in G_j .
3. By saving the path found between c_i and c_{i+1} , we obtain a characterization of the repeat that lies between the unitigs. This will not tell us exactly which reads or even which unitigs come from the repeat in question, but by examining the sequence of the unitigs along the path we will have a good idea of what the repeat looks like. This is equivalent to the definition of *sequence characterized* given in Section 8.

These searches of the repeat blizzards separated connected unitigs was performed on our test data. Before the searches were performed the graph of uniques, G , contained 18 connected components. The searches allowed two pairs of connected components to be joined, resulting in a total of 16 connected components. Before the searches the

average error in distances between adjacent unitigs was 48 bases. After the searches the average error was 7 bases. This validates point 2 listed above.

After performing these searches for all pairs of adjacent unitigs in the maximal connected subgraphs and connecting the subgraphs that are found to be interleaved, the maximal subgraphs should cover non-overlapping portions of the source. We now use the unconfirmed BAC ends and a search of the repeat blizzards off the end unitigs to connect the subgraphs. A weighted graph G' is constructed whose nodes are the ends of the maximal connected subgraphs of the graph of uniques, G . Thus if $|G| = n$, then $|G'| = 2 \times n$. Edges in this graph are determined by the BAC ends. Suppose G_i and G_j are two elements of G , and that m_1 and m_2 are mated BAC ends that lie in G_i and G_j respectively. Arbitrarily label the ends of G_i as A and B and the ends of G_j as C and D . If we assume the BAC to be good, then the orientation and position of the BAC ends within the connected subgraphs gives a distance and relative orientation of the subgraphs, for example, $AB \ CD$. In this case there would be an edge between B and C in G' . The weight of the edge would be equal to the number of BAC ends that indicated such an orientation.

The next step of the algorithm is to find a subgraph of G' such that the maximum degree of any edge is one. We proceed in a greedy fashion, taking the edge with highest weight that connects two nodes that both have a degree of zero in the current state of the subgraph. A search through the repeat blizzard off the ends of the indicated connected components from the graph of uniques is then performed. If a connecting path of acceptable length is found, then the edge is added to the subgraph.

The procedure allows us to connect many of the maximal connected subgraphs of G . Ideally, we would desire a single connected graph ordering all the unique unitigs. We ran this procedure on our test data. The result was that the 16 connected components we had left at this point were reduced to 3. One of these consisted of a relatively small (1279 bases) single unitig. The other two consisted had lengths of 261,706 and 3,229,119 bases. The final statistics on the markers were:

Number of <i>UNKNOWN</i> BAC end pairs	14
Number of <i>CONFIRMED</i> BAC end pairs	182
Number of <i>BAD</i> BAC end pairs	37

As was the case earlier, no markers from chimeric BAC's were confirmed, and no good markers were marked as *BAD*. Notice that 182 or 95% of the good markers were confirmed, which is not too far off from the idealized simulator results given in section 9.1.

11 CONCLUSION

Recent events have somewhat overtaken the work in this thesis. As stated earlier, this work is a direct follow up to the work of Weber and Myers. They demonstrated that scaffolds connecting markers almost always exist. In this work we ran the first simulations on algorithms to assemble the sequence between these markers. The simulated results are very promising. However, before they were completed, Celera announced its intention to sequence the human genome using the whole-genome shotgun approach. So, as much as I would like it, it cannot be claimed that the results given here had any influence on that decision (although the Weber and Myers work probably did). However, the scientists faced with the huge task of building the Celera assembler might well have gained a bit of comfort from these positive results.

While the Celera assembler is not utilizing these algorithms, some of the ideas developed here have been incorporated. Perhaps the most original and important idea is the identifying and isolating of the repeats through edge detection. While this seems like an obvious approach, no existing assembler does this. The TIGR assembler³⁸ does try to identify repeats by looking for a large number of overlaps. This method, however, cannot distinguish all repeats. Low copy number repeats may be missed, and many areas that are not repetitive will be erroneously treated as repetitive. Also, since a repeat edge is not looked for, one cannot attempt to span the repeat by searching for all reads with a significant overlap.

Our tests give very good results and show great promise for the success of whole-genome shotgun sequencing. There are, however, still many open problems that were not

addressed here. Those implementing a whole-genome shotgun assembler still face the following complications.

1. **Fractured Repeats:** Some repeats are mutated subsequences of some source sequence, rather than full copies. This means that instead of looking like a sequence with several branches at the ends, repetitive footprints will have branches staggered along their length. This would not affect the quick scan algorithm since it does not build repetitive footprints. With a few adjustments, the inter-marker assembly algorithm should be able to handle this case. The biggest change would be in building the repetitive footprint since when an interior edge of a repeat was found, one wouldn't know if the repeat was terminated, or if one of the branches was a continuation of the repeat. A probable solution would be to stop building when such a branch is reached. These branches will later be explored during the search for corresponding unique footprints for each occurrence of the repeat.
2. **Repeat Blizzards:** In our simulation we made the assumption that different occurrences of the same repeat were identical. In some senses this is a conservative assumption since it means there is no information to be gleaned from an individual repeat occurrence. However, our assumption greatly simplified the creation of repetitive footprints, as all reads that covered any occurrence of a repeat are collapsed into one footprint. In reality, due to local mutations, reads overlap with some occurrences of the repeat and not others. The sets of occurrences that have overlapping reads change as a function of

where in the repeat the read is. What results is a mass of conflicting edges we call a *repeat blizzard* or *repeat tangle*. One way of dealing with this would be to attempt to collapse the repeat tangle into a single footprint by doing sequence comparisons of the reads in the tangle using a more lenient scoring scheme. Another would be to search through the blizzard for unique footprints.

3. **Tandem Repeats:** As defined earlier, tandem repeat is one where multiple occurrence of a repeat lie next to each other. For large, low copy number repeats this does not cause a great problem. A branch from one end of the repeat to the other would indicate that there was at least one tandem occurrence of this repeat. To tell how many times to traverse this loop one would use the distance information of inserts and mates that span the repeat. The coverage of the repeat would also be an important indication. A more difficult case is small tandems. In this case, a very small sequence (e.g. "AT") is repeated hundreds or thousands of times. These types of repeats were not modeled in our simulations. To deal with them one clearly must use reads that overlap these repeats but also have at least *MIN_OVERLAP* unique bases on the end. These will give a minimal size for the repeat. One must then use inserts that span the repeat, coverage information, and local mutations to put the reads together correctly.
4. **Repeat Separation:** Other than the obvious step of using mate pairs to anchor reads that lie within a repeat (our notion of *uniquely assembled* from

section 8), we have not addressed the idea of repeat separation. That is, determining for each read that lies within a repeat which occurrence of the repeat it comes from. If all occurrences of each type of repeat were identical, as in our model, this would be unimportant. However, for most repeats in real genomes, each occurrence varies slightly. Separating the reads into their correct location is essential to determining the exact sequence of each occurrence of the repeat.

5. **Structural Polymorphisms:** These are differences between copies of the same chromosome. Since humans have two copies of each chromosome (except the X and Y), even if only one donor was used to do the sequencing, whole-genome shotgun sequencing must deal with structural polymorphisms. As described earlier, most differences involve a single base change (SNPs). SNPs present no problems to sequence assembly, and their discovery is one of the important benefits of the whole-genome shotgun approach. Structural polymorphisms, however, involve multiple bases. Usually these are caused by a sequence that has been moved or copied to another location. Such polymorphisms are expected to be very rare, and our simulator did not model them. When building an actual assembler for whole genome shotgunning, however, one must consider such possibilities.

Though we have demonstrated promising results on a reasonable simulation, it should be clear from what is written above that there are many problems that still need to be mastered to build a working assembler. So what are the prospects for success of

whole-genome shotgun assembly? I believe the evidence shows them to be quite good. The results of our simulations seem to show that the method is viable. None of the problems listed above seem to be insurmountable, and there is currently a dedicated team of bright scientists working to solve them. One of reasons the human genome project rejected the proposal to do whole-genome shotgun sequencing is that it was said to be unproven. Using the shotgun method to sequence a lower organism, it was claimed, was not a sufficient indication that it could be done for a creature with a more complex genome. That being said, a very good indication that this method will work is the successful sequencing of the fruit fly, *Drosophila*.³⁹ This is the most complex genome ever completely sequenced and it is a great demonstration of the feasibility of whole-genome shotgun sequencing. The successful assembly of this genome gives us every reason to believe that this method will yield the entire human genome by 2001.

12 REFERENCES

- ¹ A. Maintland. Designer Crops: Genetic Feast or Famine. *Financial Times*. January 9, 1998
- ² J. Watson and F. Crick. Genetical Implications of the Structure of Deoxyribonucleic Acid. *Nature* 171 (1953), 737-738.
- ³ A.M. Maxam and W. Gilbert. A New Method for Sequencing DNA. *Proceedings of the National Academy of Science USA* 74 (1977), 560-564
- ⁴ F. Sanger, S. Nicklen, and A.R. Coulson. DNA Sequencing with Chain-Terminating Inhibitors. *Proceedings of the National Academy of Science USA* 74 (1977), 5463-5467
- ⁵ F. Sanger, A.R. Coulson, G.F. Hong, D.F. Hill, and G.B. Petersen. Nucleotide Sequences of Bacteriophage λ DNA. *Journal of Molecular Biology* 162 (1982), 729-773
- ⁶ H. Peltola, H. Soderlund, and E. Ukkonen. SEQAID: A DNA Sequence Assembly Program Based on a Mathematical Model. *Nucleic Acids Research* 12 (1984), 307-321
- ⁷ X. Huang. A Contig Assembly Program Based on Sensitive Detection of Fragment Overlaps. *Genomics* 14, 18-25
- ⁸ S. Larson, M. Jain, E. Anson, E. Myers, An Interface for a Fragment Assembly Kernel, Tech. Rep. TR96-04, Dept. of Computer Science, Tucson, AZ (1996)
- ⁹ E. Myers. A Sublinear Algorithm for Approximate Keyword Matching. *Algorithmica* 12 (1994), 345-374
- ¹⁰ J. Turner. Approximation Algorithms for the Shortest Common Superstring Problem. *Information and Computation* 83 (1989), 1-20
- ¹¹ J. Tarhio and E. Ukkonen. A Greedy Approximation Algorithm for Constructing Shortest Common Superstrings. *Theoretical Computer Science* 57 (1988), 131-145
- ¹² A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear Approximation of Shortest Superstrings. *Journal of ACM* 41:4 (1994), 630-647
- ¹³ J. Kececioğlu and E. Myers. Exact and Approximate Algorithms for the Sequence Reconstruction Problem. *Algorithmica* 13 1-2 (1995), 7-51
- ¹⁴ R. Idury and M.S. Waterman. A New Algorithm for Shotgun Sequencing. *Journal of Computational Biology* 2:2 (1995), 291-306

-
- ¹⁵ D. Feng and R. Doolittle. Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees. *Journal of Molecular Evolution* 25 (1987), 351-360
- ¹⁶ A. Drogh, M. Brown, I.S. Mian, K. Sjolander, and D. Haussler. Hidden Markov Models in Computational Biology. *Journal of Molecular Biology* 235, 1501-1531
- ¹⁷ E. Anson and E. Myers. ReAligner: A Program for Refining DNA Sequence Multi-Alignments. *Journal of Computational Biology* 4:3, 369-383
- ¹⁸ E.S. Lander and M.S. Waterman. Genomic Mapping by Fingerprinting Random Clones: a Mathematical Analysis. *Genomics* 2 (1988), 231-239
- ¹⁹ R. Fleishmann *et al.* Whole Genome Random Sequencing and Assembly of Haemophilus Influenzae. *Science* 269 (1995), 496-512
- ²⁰ A. Edwards and C.T. Caskey. *Methods: Companion Methods in Enzymology* 3, 41 (1991)
- ²¹ F. Collins and D. Galas. A New Five-Year Plan for the U.S. Human Genome Project. *Science* 262 (1993), 4 3-46
- ²² M.R. Olsen, *et al.* Random Clone Strategy for Genomic Restriction Mapping in Yeast. *Proceedings of the National Academy of Science USA* 83 (1986), 7826-7830
- ²³ Y. Kohara, A. Akiyama, and K. Isono. The Physical Map of the E. Coli Chromosome: Application of a New Strategy for Rapid Analysis and Sorting of a Large Genomic Library. *Cell* 50, 495-508.
- ²⁴ A. Coulson, J. Sulston, S. Brenner, and J. Karn. Toward a Physical Map of the Genome of the Nematode, C. Elegans. *Proceedings of the National Academy of Science USA* 83 (1986), 7821-7825
- ²⁵ M.R. Olson, L. Hood, C. Cantor, and D. Botstein. A Common Language for Physical Mapping of the Human Genome. *Science* 245 (1989), 1434-1435
- ²⁶ F. Alizadeh, R.M. Karp, L.A. Newberg, and D.K. Weissner. Physical Mapping of Chromosomes: A Combinatorial Problem in Molecular Biology. *Algorithmica* 13:1/2 (1995), 52-76
- ²⁷ M. Jain and E. Myers. Algorithms for Computing and Integrating Physical Maps Using Unique Probes. *Journal of Computational Biology* 4:4 (1997), 449-466
- ²⁸ J. Weber and E. Myers. Human Whole Genome Shotgun Sequencing. *Genomic Research* 7 (1997), 401-409

-
- ²⁹ G. Myers and J. Weber. Is Whole Genome Sequencing Feasible? in *Computational Methods in Genome Research* (S. Suhai, ed.) Plenum Press (New York, 1997), 73-89
- ³⁰ P. Green. Against a Whole-Genome Shotgun. *Genomic Research* 7 (1997), 410-417
- ³¹ T.J. Hudson, *et al.* An STS-Based Map of the Human Genome. *Science* 270 (1995), 1945-1954
- ³² M. Olson, L. Hood, C. Cantor, and D. Botstein. A Common Language for Physical Mapping of the Human Genome. *Science* 245 (1989), 1434-1435
- ³³ J.C. Venter, M.D. Adams, G.G. Sutton, A.R. Kerlavage, H.O. Smith, and M. Hunkapiller. Shotgun Sequencing of the Human Genome. *Science* 280 (1998), 1540-1542
- ³⁴ F.S. Collins, *et al.* New Goals for the U.S. Human Genome Project: 1998-2003. *Science* 282 (1998), 682-689
- ³⁵ E. Myers, *et al.* A Whole-Genome Assembly of *Drosophila*. *Science* 287 (2000), 2196-2203
- ³⁶ E. Myers. Toward Simplifying and Accurately Formulating Fragment Assembly. *Journal of Computational Biology* 2:275 (1995), 275-290
- ³⁷ E. Myers, A Data Set Generator for Whole Genome Shotgun Sequencing. *Conference of Intelligent Systems for Molecular Biology*, Heidelberg, Germany, (1999) 202-210
- ³⁸ G.G. Sutton, O. White, M.D. Adams, and A.R. Kerlavage. TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Science and Technology* 1:1, 9-19
- ³⁹ M. Adams, *et al.* The Genome Sequence of *Drosophila melanogaster*. *Science* 287 (2000), 2185-2195