ABSTRACT

Title of dissertation: **IMPROVING GENOME ASSEMBLY**

Cevat Ustun, Doctor of Philosophy, 2005

Dissertation directed by: Professor Jim Yorke and Brian Hunt
Department of Physics and Math

We present a reliable, easy to implement algorithm to generate a set of highly reliable overlaps based on identifying repeat k-mers. Our method is coverage independent. Whereas traditionally reads have been trimmed to have expected error rates of 2%, we find our error correction allows extending usable sequence in reads to 16% trimming. We use a version of the *Phrap* assembly program that uses only overlaps computed by the UMD overlapper, called *PhrapUMD*. We integrate the UMD algorithms with Baylor's ATLAS assembler applied to *Rattus norvegicus*. Starting with the same data as the Nov. 2002 ATLAS assembly, we compare our results to 4.5 Mbp of rat sequence in 21 BACs that have been finished. We find that after extension and error correction, (i) the reads are 30% longer than reads trimmed to 2%; (ii) the average error rate across the extended reads is about 3 in 10,000 bases, with 88% of the extended reads matching finished sequence *exactly* across their *entire* length; and (iii) PhrapUMD with these reads and our reliable overlaps produces a draft assembly of the rat which has no misassemblies and increases the coverage of finished sequence from 92.2% to 95.7%, while simultaneously reducing the base error rate for quality 20 or higher bases from 1.50 to 0.87 errors per 10,000.

IMPROVING GENOME ASSEMBLY

by

Cevat Ustun

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

Advisory Commmittee:

Professor Jim Yorke, Advisor
Professor Brian Hunt, Chair/Co-Advisor
Professor Edward Ott
Professor Stephen Altschul
Professor Wolfgang Losert

© Copyright by

Cevat Ustun

2005

FOREWORD

The discovery of the structure of deoxyribonucleic acid (DNA) in 1953 by Watson and Crick promised a revolution in our understanding of the mechanics of living organisms. What they found was that this structure consisted of four types of molecules (or bases), referred simply by their initials A, C, G, and T, and that these bases were arranged in two long strands held together by weak molecular bonds. This results in what is commonly referred to as the double-helix. Furthermore, it was found that the progression of bases along a *single* strand is arbitrary but the corresponding base on the complementary strand at a given point along the double helix are constrained; an A can only match up with a T, and a C only with a G (in what follows, we will consider only a single such strand).

The determination of the sequence of bases for a particular organism would then (in theory) only involve reading off the consecutive bases from a given strand of DNA. Simple as it sounds, it would be some years before the determination of actual genomes of even the simplest of organisms were attempted.

The current technology for the determination of bases in a sample of DNA, due to Sanger, involves doing a rather involved experiment in which the accuracy of what is read drops sharply after roughly 1000 bases (which we will write as 1000 b) from either end of a DNA fragment. The parts of DNA which have been sequenced this way are known as "reads" and reads originating from the opposite ends of

the same fragment are referred to as "mate pairs". By itself, this method solves little since genome sizes of organisms are many times the read size: at the lower end, mitochondrial DNA is on the order of 10,000 base pairs (i.e., 10 kb) whereas towards the high end, for organisms including humans, the figure is around 3 billion base pairs.

The standard way around this problem is called shotgun sequencing and it starts off by shattering multiple copies of the entire DNA in question and then sequencing the ends of the resulting small fragments without knowing where the fragments came from in the genome. The goal then is to identify overlapping reads on the basis of their sequence and merge them to obtain the full genomic sequence. In practice this approach yields a "draft" of the genome that is incomplete and riddled with errors because of a number of complicating factors. Among the most prominent are:

i) The coverage of the genome by reads in the shotgun sequencing approach is not uniform; in fact, the coverage is well described through most of the genome in terms of a Poisson statistic. This means for example that for any given coverage, there is an appreciable probability of finding gaps in the genome not represented by any read and also for finding an unusually high number of reads covering a particular area.

ii) The sequence of most organisms contain "repeats". These are sections of the genome usually up to about 10 kb long that have exact or approximate copies of themselves elsewhere in the genome. The number of copies can run up into the hundreds. Repeats will typically cause an assembly software to place reads originat-

ing from different parts of the genome at a single spot because of the similarity in the sequence. Such mis-assemblies may not be easy to detect because of (i) above. Differences between copies of repeats (due to mutations) can at times be used to distinguish them from one another.

iii) Sequencing errors occur when resolving the bases in a read. Such errors also make the job of separating slightly different copies of repeats harder as in (ii) above.

It is the interaction of the above difficulties that make this a non-trivial task. Despite the years of refinements in sequencing techniques and assembly methods, the above factors can still confound assemblers, adding significantly to the already enormous costs of sequencing genomes. It is an ongoing effort to develop methods for getting more information, that is more accurate, more complete drafts of genomes.

ACKNOWLEDGMENTS

A work such as a dissertation is not possible without the guidance, support and insight of many people. I owe a tremendous amount of gratitude, first and foremost, to my advisers Jim Yorke and Brian Hunt. It is their voice that resonates within me whenever I'm at a loss of what to.

Thanks are also due to my co-workers, Aleksey Zimin and Mike Roberts. Mike deserves special thanks for all the suffering he has had to endure in providing me most of the data that has been crucial for this work.

Lastly, I would like to thank my loving family, from whom I received nothing but encouragement and support in my years of graduate study.

Thank you all!

TABLE OF CONTENTS

Chapter 1

Improving Sequence Assembly of the Rat Genome Using Reliable Overlaps and Extended Reads

1.1 Introduction

Improving WGS. Whole genome shotgun (WGS) assembly has created draft versions of several large genomes, including the rat *Rattus norvegicus*. However, to our knowledge, no one claims that these methods have been refined to the point where they are the best possible. While it is not known how much room there is for improvement of existing assemblers, it is clear that the majority of the cost of assembly is the creation of read data. The Baylor College of Medicine (Baylor) assembler ATLAS already used innovative techniques to assemble the rat genome. The University of Maryland authors (UMD) are collaborating with Baylor to identify possible ways of getting even better draft assemblies from the data. As a test case, the rat genome is being reassembled using UMD modules within ATLAS. For this assembly we are using the same read and mate-pair data (called "Freeze 04") that was used to produce the most recently published draft assembly [reference paper and sequence].

Evaluation and results. To evaluate our techniques, we compare our assemblies of 21 BACs that lie in the small fraction of rat sequence that is of finished

quality. The finished sequence has been upgraded through intensive additional sequencing efforts so that evaluation is possible. However, in order to make their final assembly of high enough quality to be useful for biologists, Baylor used all available data to produce Freeze 04 assembly including the reads used to finish that sequence. Therefore, to evaluate the addition of UMD techniques to ATLAS, we return to the so-called "Freeze 02" assembly, which did not include reads from finishing efforts. Based on a comparison with the assemblies of 21 finished BACs, for 12% extended and retrimmed data, we recover 3.5% additional sequence matching to the finished sequence at almost half the overall error rate for bases with quality 20 or higher.

ATLAS. To give the flavor of the ATLAS strategy for rat assembly, we summarize its procedures as follows. See [ ] for more details. Baylor first broke the genome into about 20,000 carefully selected, overlapping BACs. The Rat Genome Sequencing Consortium (RGSC) produced whole genome shotgun (WGS) reads and also lightly sequenced each of these BACs. ATLAS estimated which WGS reads overlapped the BAC reads of a given BAC and added those WGS reads and their mates to the BAC's reads to create a "bin" of reads for the BAC. ATLAS then applied Phrap [ref] to the binned reads to produce an assembly of the BAC. Then ATLAS created a rat assembly by piecing these BAC assemblies together, using mate pair information to correct many of the errors that Phrap makes.

UMD+ATLAS. The UMD+ATLAS results were obtained by incorporating three UMD sets of techniques into ATLAS.

- The UMD overlapper [published paper] corrects errors in reads and produces

overlaps with a goal of missing at most one true overlap in 10^8 (with overlap length of at least 40 bases).

- We used the power of (1) to trim the reads much less aggressively here, thereby allowing longer reads. Traditionally reads are trimmed when the quality of the bases indicates an error rate of about 2%. We trimmed only when expected error rate reached 8% to 16%.

- We developed algorithms to find a subset of the above overlaps that we call "reliable" for use in generating an assembly. Using only reliable overlaps in creating assemblies produces more sequence with fewer errors. An occasional contig will broken in two if only reliable overlaps are used in the assembly, so we sometimes use other overlaps from the UMD overlapper list to join contigs that ATLAS says are adjacent.

PhrapUMD. En route to creating a rat assembly, we developed additional techniques to help ATLAS benefit from extended reads and reliable overlaps, including alterations of Phrap to force it to use only overlaps from a list it is given. We call the modified program PhrapUMD. ATLAS's use of Phrap allowed the use of UMD high-quality overlap information only for the binning process. PhrapUMD now assures that assemblies use only the overlaps we choose.

Baylor's overlap seeds. The criterion in (3) for creating reliable overlaps was motivated in part by Baylor's approach. ATLAS was already using innovative and effective overlap identification technique that is quite different from those used by other assemblers. WGS assemblers begin by examining small read fragments,

called *k-mers*, usually consisting of 20 to 32 bases (20 bases for this paper). These are also referred to as "seeds". When reads are found to have a seed in common, it is possible to see if the match can be extended to yield a plausible overlap. We say the seed "generates" the overlap. While Celera's version of the seed-and-extend step strongly emphasized finding all plausible overlaps, ATLAS took the opposite route. It only used a seed that occurred sufficiently rarely in the database of reads (12 times or less) that it appeared likely to come from a unique place in the genome. Many spurious overlaps are thereby avoided. A seed coming from multiple places in the genome will generally occur more than 12 times in the read database.

Reliable overlaps. We classify overlaps as reliable by using a more direct approach to identifying seeds from repeat regions. The UMD overlapper [ref] creates a lean list of overlaps while missing extremely few correct overlaps. Of necessity some overlaps of questionable quality are left in the list and some of these in fact are spurious, but if two reads overlap by at least 40 bases, they are virtually certain to be in the list. If two reads that belong together, but are not reported as overlapping, then any seeds they have in common almost certainly lie in repeat regions. We create a list all seeds that belong to repeat regions. An UMD overlap that has no discrepancies is called "reliable" if the two reads have at least one seed in common that is not in the list of repetitive seeds.

This paper is organized as follows. Section 2 provides the description of the algorithm that produces a subset of reliable UMD overlaps. Section 3 describes the process of read extension and error correction and section 4 presents the assembly results. We provide the technical details and the detailed description of the assembly

pipeline, as well as assembly evaluation process and PhrapUMD design in appendices A-D.

1.2 Methods

1.2.1 Reliable overlaps

To identify the plausible overlaps we first look at all k-mers ($k = 20$ in out methods) in the reads and pick up the subset of them that we call "minimizers". The concept of minimizers is described in detail in Appendix A. This method allows us to choose about 10% of the possible k-mers in a given read as minimizers, so that they have the following properties:

- (i) The collection of minimizers cover the entire read, except possibly for a few bases at the ends of the read; and

- (ii) If two reads have significant exact overlap, then many of the minimizers chosen from one will also have been chosen for the other.

On average, one out of 10 consecutive 20-mers along the read sequence is a minimizer, in other words, we encounter a new minimizer on average every 10 bases. If two reads overlap by at least 40 bases, they are guaranteed to have at least one minimizer in common.

We obtained results reported in this paper by using an extremely simple method of identifying *repeat minimizers*, that is minimizers that are likely to belong

to repeat regions. Of course, we could have looked at the frequencies of the occurrence of all minimizers to distinguish repeat from non-repeat ones, but this would require imposing a cutoff, which would depend on the coverage and it would not work if coverage varies greatly along the genome sequence, as it is the case for Rat data. Our goal was to design a method that is easy to implement, that has no tunable parameters, and that could be applied to any data set without modifications.

Our method is the following. We first examine all overlapping read triples. Let us pick three reads, A, B and C and assume that read A has reported overlap with read B, and read B has reported overlap with read C, and based on the offset information, reads A and C should overlap, but they are not reported overlapping. This situation is very common: it occurs where part or whole read B is in a repeat region and reads A and C represent two possible exits from the repeat. In this case we look at reads A and C and add all common minimizers in reads A and C to the list of repeat minimizers. See Fig. a for illustration. Then we examine all overlaps and for each overlap find the number of the common reliable (non-repeat) minimizers. We declare an overlap to be reliable if the number of common reliable minimizers is greater than the number of differences in the overlap region multiplied by two (case with no differences is shown on Fig. b. The latter requirement is imposed to avoid declaring reliable repeat-induced overlaps that have errors in the read sequences. We multiply the number of differences by two because on average each base is covered with two minimizers, so each error will usually influence the values of two minimizers. This is a very simple and easy to implement method, and it produces excellent results described in Section 4.

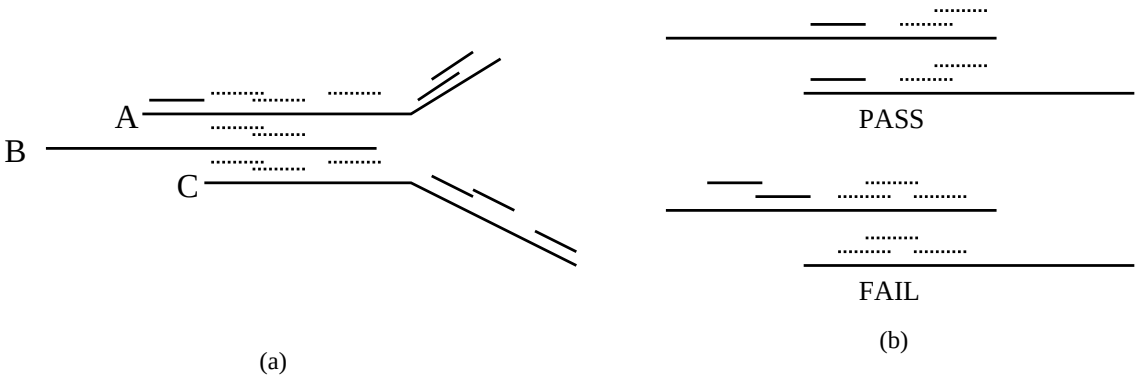

Figure 1.1: Reliable minimizers are represented as solid lines and unreliable ones are represented as dashed lines. (a) identifying reliable minimizers; (b) reliable overlap test.

1.2.2 Read Extension and error correction

Traditionally, excessive sequencing errors have tended to confound assembly. Thus traditionally reads are trimmed such that over any local region the error rate does not exceed 2%. This results in using about half of the sequence available in the untrimmed read. Since sequence is expensive to produce, we would like to use more of the available sequence. Our approach to this problem is to use our extremely high-quality overlaps [32] to create multi-alignments, which allow us to correct a large number of low-quality bases well beyond the 2% trimming mark. We also detect and mark questionable bases that we are unable to correct. This very reliable yet conservative error correction allows us to extend reads out to trimmings of up to 16%, while making very few erroneous corrections.

After trimming a read to X%, we correct a base if it satisfies either of the following two cases (unless it satisfies one of the exceptions listed afterwards):

Case 1: There are at least two overlapping reads at the base's position, and there is unanimous consent among them that the base is wrong, **or**

Case 2: "Unanimous−1" consent, in which we require the following criteria to be met:

(a) at least 4 overlaps cover the base (i.e., 5 reads including the one being corrected) with all but one agreeing to the change, **and**

(b) there is at least one minimizer covering the base, and all the minimizer counts for minimizers covering the base are 1.

This criterion is designed to ensure that the base we want to correct has absolutely no corroborating evidence to support its current value. Furthermore, statistical tests have shown that this corrects significantly more bases than it destroys.

Case 2 is designed to avoid changing regions with "low" (but more than 1X) coverage. For example, assume the questionable base occurs in a repeat region that occurs twice; one of them with 3X coverage and the other with 2X coverage, and our base is in the 2X coverage region. This example satisfies Case 2, where we would observe a 5X region where the base "vote" is 3-2; Case 2b is invoked, refusing to make any change to the 2X coverage region. See Figure 1.2.

Unfortunately, even within these cases, we can potentially make incorrect changes to 1X coverage bases. In an attempt to avoid this, we add a "catch-all" rule based

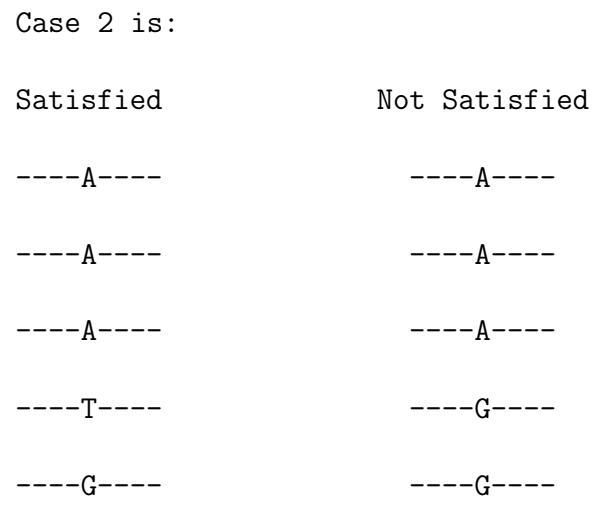

Figure 1.2: Demonstration of Case 2. We are trying to decide whether or not to change the base 'G' in the bottom read. Places marked with a '-' agree in all reads. On the left, we have unanimous - 1 consent and so we change the 'G' to an 'A'. On the right, the second 'G' corroborates the first, and so we do not make the change, according to Case 2b.

on the rationale that it would be rash to correct several high-quality bases. Thus, before we actually make the changes for either case, we look at the quality values of all the bases over the suggested changes, and sum together each quality that is greater than 30. If the resulting sum is 70 or greater, we make *no* changes to the read. This is to avoid using spurious overlaps with repeat regions to change high-quality bases in a 1X coverage region. Quality 30 is simply the observed quality of an average, reasonable base. The 70 is *ad-hoc*, but based on the idea that if we are trying to change more than 2 high-quality bases, then the read may come from a high-quality 1X coverage region. This rule could possibly be eliminated or weakened if we wanted to correct more errors at the expense of introducing more incorrect changes.

Since minimizers do not always cover the ends of a read, we can miss error corrections there due to Case 2. However, the same base may be corrected in Case 1 on later iterations because the consent may become unanimous due to corrections to the one non-unanimous read.

Obviously, we do not want to use spurious overlaps to compute corrections. We use a Poisson test to "pass" each overlap so that a true overlap has a probability of being rejected of only 10^{-8}. This allows a significant number of spurious overlaps to be passed, which is acceptable for assembly purposes (because other criteria will be used to weed out spurious overlaps later), but is unacceptable for error correction purposes at such an early stage in assembly. We have found that if we reject at the 10^{-2} level rather than at 10^{-8}, then the number of spurious overlaps is negligible, and so we reject the bottom 1% of our overlaps for error correction

purposes. Furthermore, at the 10^{-8} level, there would be situations where it would be difficult to get the (nearly) unanimous consent we need to correct bases, due to the presence of spurious overlaps. We consider this to be an extremely conservative method of error correction.

Retrimming excessive errors off the ends of extended reads

To recap, we start with reads that are trimmed (before correction) to a rate of 2%, and then run our overlapper and error-correction routines, as described in [32], yielding substantially longer reads. Now, these corrected reads have base error rates which are substantially improved from the original reads. However, the error rate in the original reads increases rapidly near the back end of the read, and at some point the errors are so frequent that we are unable to reliably correct them. The task now is to detect the end of our reliably corrected region.

Recall that to correct a base in read R_0 at position p requires an almost unanimous consent between overlapping reads at p. Error *detection*, however, requires less information than error *correction*; in particular, R_0 may still disagree with a *minority* of overlapping reads at several positions. We now have two competing goals: (1) to detect and trim excessive errors from the inside end of R_0, and (2) avoid excessively trimming good sequence in R_0 based on disagreements with spurious overlaps – which are rare in our overlap sets but do occur. To reconcile these two goals, we compare R_0 to each overlapping read in turn, and then trim R_0 based on the overlapping read which agrees *best* with R_0, based on the assumption that this

is the read that is most likely to truly overlap with R_0 and thus disagreements with it are most likely to indicate actual errors in R_0. Let R_i be any overlapping read with R_0. We record each disagreement between reads R_0 and R_i. We then choose the largest trim position (numbering from the beginning of the read) so that there are no more than 5% of the bases are in disagreement for *any* window starting from a potential trim point extending towards the beginning of the read. (We aim for a maximum error rate of 5% after correction because this is the error rate assumed by Phrap during Baylor's rat assembly.) If the resulting overlap between R_0 and R_i is less than 60 bases, then we do not consider this overlap for the trimming calculation. Finally, after performing the above procedure individually for each read R_i that overlaps R_0, we choose the trim position that gives the *longest* read R_0. Note that if all the resulting trimmed overlap lengths are less than 60 bases, then the resulting read is eliminated from the assembly.

Table 1.1 presents the average read lengths obtained as a function of trimming, with and without retrimming, across the 21 BACs listed in Figure B.1. As expected, the extended and retrimmed reads get longer as we extend to higher trimming values.

We measured total (insertion/deletion+substitution) errors across all of the 12% trimmed, corrected, and re-trimmed BAC reads in the bins of our 21 BACs by finding the best match for each BAC read in its appropriate BAC and counting base differences. (We did this only for BAC reads because we did not want to contaminate the results with incorrectly binned WGS reads.) Figure 1.2 lists the number of reads with a given total number of individual base errors. Almost 13,000 out of 14,700 BAC reads, or 88.1% of them, have *no errors*, and match the finished

% trimming	avg. length	avg. retrimmed length
2	564.9	
8	624.0	613.5
10	643.8	631.0
12	665.3	648.1
16	734.2	712.7

Table 1.1: Average read lengths as a function of trimming percentage, before and after retrimming.

sequence exactly; 96.1% have at most one base error; and 98.1% have at most 2 base errors; and the total number of base errors is 2,971. The total number of bases in these 14,720 reads is 9,877,120, and 2,971/9,877,120 evaluates to roughly 3 errors per 10,000 bases.

1.3 Results

All results reported in this section are for the set of 33 million *Rattus norvegicus* reads available in November 2002. The average read coverage of our 21 BACs is about 7.0 for our 2% trimmed and corrected reads (31469211 bases in reads across 4504811 bases of finished sequence) For the entire 33 million reads average coverage is about 7.3.

ATLAS uses mate-pair information to order and orient the Phrap-built contigs, as well as using this information to detect and re-assemble contigs that Phrap appears to have misassembled. ATLAS then assigns each such scaffold an internally

#errs	#reads	cum#reads	cum%reads	cum#errs
0	12974	12974	88.1%	0
1	1173	14147	96.1%	1173
2	298	14445	98.1%	1769
3	98	14543	98.8%	2063
4	62	14605	99.2%	2311
5	49	14654	99.6%	2556
6	47	14701	99.9%	2838
7	19	14720	100%	2971
total	14720	14720		2971

Table 1.2: Histogram of individual base errors (indels+substitions) compared to finished sequence, for reads trimmed to 12%, then error corrected, and retrimmed to delete excessive errors.

computed quality value. Those with too low a quality are discarded by ATLAS. In this analysis, we will consider only the high-quality, non-discarded scaffolds.

We perform the analysis on the binned BAC reads and then use PhrapUMD and ATLAS to create an assembly based on the set of reliable overlaps. We match the resulting assembly to the finished sequence. The exact criteria and methods for matching the assemblies are presented in the Appendix C. We measure the following quantities for each assembly:

- Non-Matching Contig Tails: the total number of bases in tails of Xcontigs that match finished sequence.
- Non-Matching Contigs: the total number of bases in Xcontigs larger than 1k that do not match finished sequence in their appropriate BAC.
- Finished Sequence Matched Uniquely: the number of bases of finished sequence that are covered by at least one matching base in a matching Xcontig.
- Total Sequence Matched: the number of bases in Xcontigs that match finished sequence. This number is greater than or equal to the Unique Sequenced Matched, and is greater only when Xcontigs overlap.
- Total Sequence Matched Multiple: the number of bases in Xcontigs that overlap each other when matched to finished sequence; these are Xcontigs that probably could have been merged, but which failed to be merged in the assembly.
- % of Finished Sequence Matched: the percentage of the span of finished se-

quence of a BAC that is matched by Xcontigs longer than 1k. Erroneous bases are counted in this number. If a finished base is matched by more than one Xcontig, the base is counted only once.

- Errors per 10000 Bases: the number of bases with Quality 20 or more per 10,000 in matching Xcontigs that do not match finished sequence (substitutions, insertions, and deletions). If a finished base is covered by more than one Xcontig, then every incorrect Xcontig base is counted.

- Number of Contigs: total number of contigs in the 21 BACs.

The results for 21 BACs and for various read extension/retrimming tolerances are presented in Table 1.3. We used only reliable overlaps defined in Section 2 for this assembly. Note that the number of contigs is still large compared with Baylor assembly. Another problem is that the amount of sequence matching multiple is relatively high – this indicates that the ends of the contigs may overlap in many cases. The sequence that we obtain using the overlaps rated by this method is close to finished sequence quality.

To reduce the number of contigs and improve the multiply-matching sequence numbers, we used a method of contig merging, described in the Appendix C. Table 1.3 presents the results after the contig merging. In this method we merged the contigs when their ends overlapped according to the list of all UMD overlaps. If the ends overlapped, we added the overlaps between the end reads to the list of CONTIG overlaps, and then we reassembled the data using the set of reliable overlaps appended with the set of CONTIG overlaps. Note that as the number of contigs

decreased, the amount of sequence matching multiple decreased as well, and overall sequence quality improved. After contig merging we have exactly one scaffold per BAC for all 21 BACs.

As we increase the read extension from 2% to 12% we notice an improvement in the number of contigs and total span of finished sequence matched with simultaneous decrease in the number of non-matching contig tails. At 16% the numbers turn not in favor, so based on this data set we conclude from Table 1.3 that using 12% extension is optimal for assembly purposes. We are now using the latest data in Freeze 04 data set with to produce complete draft Rat genome assembly with 12% extended reads. We will post the assembly on NCBI website as soon as it is finished.

Another way to evaluate the quality of the assembly is to match the reads in contigs in the assembly to the finished sequence. Figure 4 shows the worst case for a particular BAC. In our assembly one notices that there are no misassemblies, the order of contigs is correct and there are no major gaps. There are two reads on the ends of two contigs that are misplaced, possibly due to repeat regions. Finishing our assembly may require much less effort and money, and the additional cost of creating our assembly is negligible.

%ext	Non-Matching Contig Tails	Non-Matching Contigs	Finished Sequence Matched Uniquely	Total Sequence Matched	Total Sequence Matched Multiple	% of Finished Sequence Matched	Errors per 10000 bases	Number of contigs
02	4020	7037	4227957	4306934	45924	95.368	0.903	587
08	3608	4960*	4231915	4304515	38399	95.457	0.685*	497
10	3214	5224	4239881	4323275	48983	95.637	0.907	501
12	2360*	7201	4245447*	4338265*	56993	95.762*	0.904	504
16	2460	25212	4203623	4296214	59697	94.819	1.194	548
ATLAS	2823	18029	4088137	4122173	8239*	92.214	1.499	377*

Table 1.3: Assembly results obtained using only reliable overlaps. Numbers that are the best in a column are marked with a *.

%ext	Non-Matching Contig Tails	Non-Matching Contigs	Finished Sequence Matched Uniquely	Total Sequence Matched	Total Sequence Matched Multiple	% of Finished Sequence Matched	Errors per 10000 bases	Number of contigs
02	7644	7037	4231017	4276779	12788	95.437	0.823	482
08	7537	4960	4224675	4269951	11139	95.294	0.656*	409
10	5007	3947*	4238768	4289247	16126	95.612	0.816	402
12	4143	9271	4242047*	4294429*	15815	95.686*	0.871	398
16	2317*	16204	4212975	4268374	21907	95.030	0.993	446
ATLAS	2823	18029	4088137	4122173	8239*	92.214	1.499	377*

Table 1.4: UMD+ATLAS assembly results obtained using reliable overlaps and contig-merging overlaps compared with finished sequence. Numbers that are the best in a column are marked with a *.

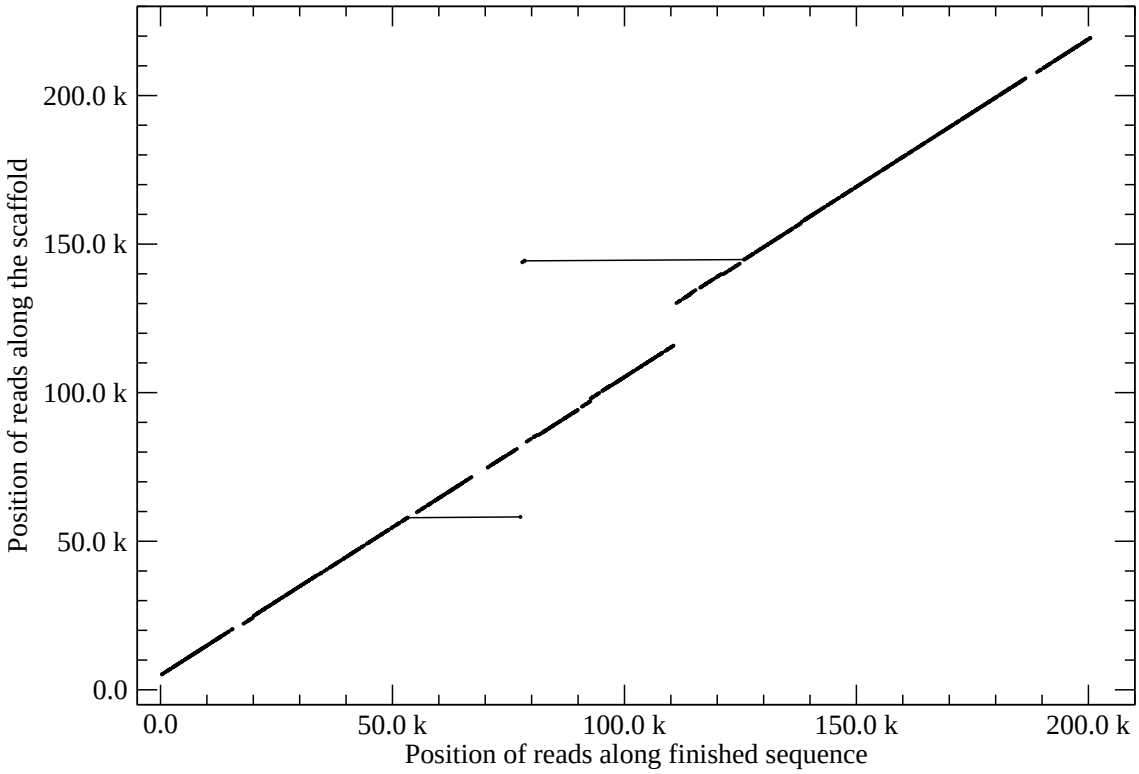

Figure 1.3: The worst case assembly is for BAC GXFC. The individual reads are depicted as dots with their place in the scaffold as their vertical coordinate and their place in the finished sequence as horizontal coordinate. Reads within contigs are connected with a line. We used the best match according to Blastz to place the reads.

Chapter 2

Constructing a physical map from whole genome shotgun data

2.1 Introduction and motivation

In 1995, Olson described a new or modified method of finding and reporting markers throughout a genome. When this procedure works well, it enables researchers to find a so-called tiling set of BACs, that is, a collection of BACs which together cover all or almost all of the genome and whose relative positions are known in the genome. Two consecutive BACs in this collection should overlap and the overlap should be quite small.

This paper will show that such a tiling set of BACs can be constructed using only WGS data without the need for expensive laboratory procedures if a combination of BAC sized and smaller sized (for example 10kbp) insert libraries (with sequenced ends) are available. We have chosen to develop our software with a simulated set of reads derived from the genome of Celegans, chosen because of its challenging repeat structure.

We first present an overview of our approach. This is followed by the characteristics of the simulated genome we have developed our software with and a discussion of the basic requirements for genomes for the methods in this paper to be applicable. A section describing the technical aspects of the algorithms used is then followed by results obtained for the simulated Celegans genome.

2.2 Overview of our approach

Two Point Boundary Problems and the Shooting Method. No pun on "shotgun" is intended. The concept of a "shooting method" is standard in differential equations and dynamical systems. Constraint information is provided at two different points, a starting point and a target point. There the goal is to find a *path* (or trajectory) running between the points. Here a "chain" is a collection of inserts, $P_1, P_2, ..., P_n$ such that each P_i has a read that overlaps a read of P_{i+1} for $1 \leq i < n$, and the orientations of the overlapping reads are opposite, as shown in Figure 1. The chain is said to **span** the insert J. Our initial goal is to find chains of small inserts spanning each large insert. The chain goes from the read at one end of the large insert, the starting point, to the other end. Of course we use estimates of the length of the short and long inserts to set a maximum and minimum allowable chain length.

Building a neighborhood structure. The shooting method allows us to find out which 10K inserts belong to chains spanning which BACs. We do not actually create all these chains to determine this. There will still be spurious overlaps so we must use this information carefully. We now invert the process. Any chain acts like a local coordinate patch. The problem of building a global map is one of putting together these local coordinate patches to create the global map, a standard concept in dynamical systems on surfaces. In this case, the surface is one dimensional.

2.3 Requirements for data and the generation of faux reads

We have developed the techniques in this paper using faux data of the Celegans genome.

2.3.1 Requirements

For our assembly technique to be applicable, we have to impose three requirements on the shotgun data. The requirements are as follows.

- **Insert size requirement.** We require that at least two libraries of mate pairs with insert sizes that differ by a factor of 4 to 20 are provided. They may be, for example, BACs (120-200Kbp long) and 10Kbp mate pairs. Ideally, we prefer 3 or more libraries.

- **Coverage requirement.** We require at least 10 times insert coverage by the biggest mate pairs (BACs) and at least 2.5 times end sequence coverage by smaller mate pairs from all other libraries.

- **Read length requirement.** We require that for each insert in the libraries mentioned above we are given both end sequences (reads) with average read length of at least 400 bases.

These requirements have been determined after extensive experiments with real-quality faux reads from *C. elegans* genome. At the present time most sequencing projects produce or intend to produce data that easily meets our requirements.

Rat assembly at Baylor For example, for rat genome the following libraries are available: a library of BACs with average insert size of 186Kbp, a library of 10Kbp mate pairs, a library of 2.5-5Kbp mate pairs and some 50Kbp mate pairs. BAC insert coverage is 11.9 times (database CHORI-230). The intended coverage by 10Kbp mate pair ends is 2.9 times and coverage by 2.5-5Kbp mate pair ends is 10.7. We expect this dataset to be close to ideal for our assembler. The proprietary Celera read library for *Drosophila* also satisfies our requirements as well as the public read library for the mouse. The rat data has the additional advantage that many BACs have been lightly shotgun sequenced (1x to 1.5x coverage).

2.3.2 Overlap experiments with *C. elegans* using faux reads

The *C. elegans* genome is known to be particularly hard to assemble, due to its rich repeat structure. It was originally put together by the method of BAC-by-BAC sequencing [6, 7, 8]. We used the most up-to-date sequence available.

The *C. elegans* results in this paper were obtained using a library of faux reads we created giving 6 times coverage of the genome. Starting with the genome we generated the faux reads with errors in the bases using a method very similar to the rather realistic approach of the Whitehead Institute group, as reported in [4].

We have generated 2.8x coverage by the ends of 10Kbp-long mate pairs, 2.8x coverage by the ends of 2Kbp-long mated pairs, 0.06x coverage by the ends of 50 Kbp-long mate pairs and 0.06x coverage by the BAC ends. We introduced a variation into the insert sizes so that the standard deviation of length was 10%. The resulting

database contained 1,065,846 reads with an average length of 537 bases. The overall error rate for bases was 0.86%. We use faux reads because with actual reads it is impossible to be certain where the errors are and which reads actually overlap. (Faux reads also have some problems, which are described in some detail in [4].)

As mentioned earlier, the reads have errors in their sequences. We did not mention but it is true that when the reads are created, each letter in each sequence is automatically assigned a "quality" value which estimates the probability that an error occurred in determining that letter. We use this "quality" data in our overlap procedures.

We can (sometimes) evaluate a list of plausible overlaps using two numbers:

T_{ratio} = fraction of all true overlaps that are in the list

F/T = false/true = the ratio of spurious overlaps in the list to true overlaps

We determine which pairs of reads overlap with very few spurious overlaps. We are able to use roughly 50% more sequence data, since we can deal with higher error rates, obtaining $F/T = 7.6\%$ and $T_{ratio} = 99.5\%$ with our *C. elegans* read library. See the right-hand column of Table 3. We do this in several stages using sequence and quality data.

2.4 Validation and the chain assembly technique

2.4.1 Validation of mated pairs

Let us begin with more precisely defining terminology used in this section. A **chain** is a sequence of mate pairs of reads, $P_i = (L_i, R_i)$, for $i = 1, ..., k$, where read

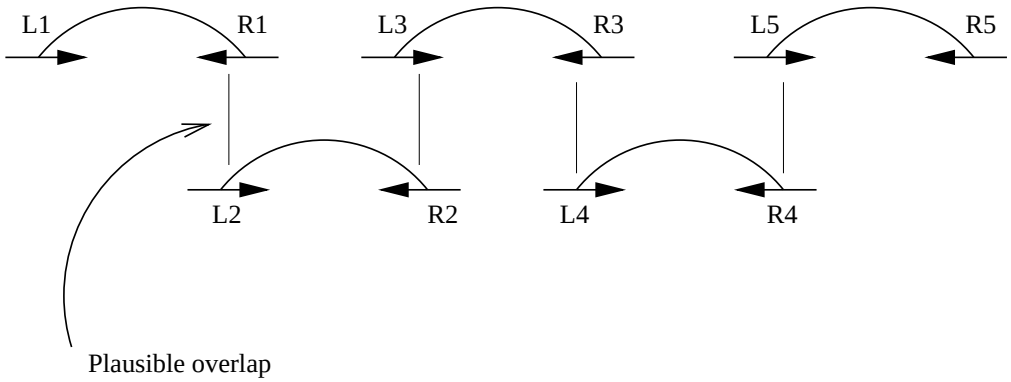

Figure 2.1: A progression of inserts making up a *chain* using plausible overlaps. Reads are denoted by the arrows indicating the direction in which the read has been read, and the dashed line connecting them is the unsequenced part of an insert. We indicate that two reads overlap by placing one above the other.

R_i plausibly overlaps read L_{i+1}. Recall that a **mate pair** consists of two reads from the ends of an insert of approximately known length (2Kbp, 10Kbp, etc.) If a chain spans a BAC, that is, L_1 plausibly overlaps with a BAC end and there exist a chain of length k for which R_k plausibly overlaps with the other end of the same BAC, such chain is said to **span** the BAC. If the calculated length of the chain is within the combined uncertainty of the mate pair and BAC length, we call such chain a **valid chain**. Each mate pair in the chain becomes **validated** by this BAC.

Since an overlap may be spurious, chains can link parts of the genome that are far apart. Starting from one read, the number of chains increases exponentially fast due to multiple coverage and repeat regions. Specifying a target (the other end of a large insert) greatly reduces that problem. This process to some extent validates the overlaps in the chain. It also validates the long mate pair, since some fraction of the inserts yield chimeric mate pairs. It is unlikely that a chain of the appropriate

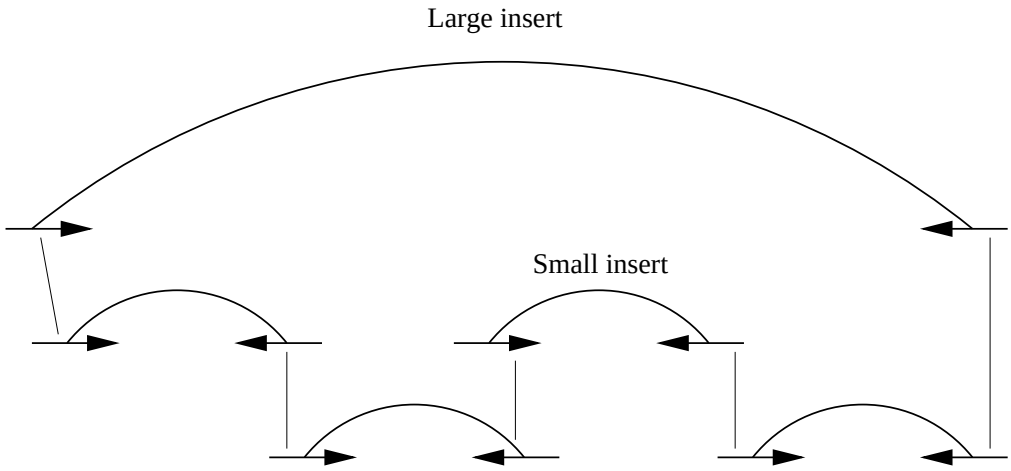

Figure 2.2: Four inserts constituting a *chain* that spans a larger insert; The collection (P_1, P_2, P_3, P_4) of small inserts is such a chain, which we call P. The left reads of P_1 and the big insert, J, overlap and have the same orientation. Similarly the right reads of J and P_4 overlap and have the same orientation. Under these conditions, we say the chain P *spans* the insert J.

length would span a chimeric pair.

2.4.2 Practical considerations

IZZn practice, we do not actually store all possible chains since there are too many of them. Instead, we start from one end of a BAC and follow the "wavefront" of all the mate pairs (from a given library) that extend in the direction of the other BAC end. Every mate pair that is used in this construction is assigned a distance from the originating BAC end based on the nominal insert sizes (2k, 10k etc) of the previous mate pairs that have been used to get to it. A similar "wavefront" is started from the opposite end of the same BAC and the mate pairs that are used

in both directions are then considered for validation. Provided that the wavefronts are extended far enough (see below), the result is a collection of mate pairs that are equivalent to those obtained by considering all chains spanning a BAC.

As mentioned above, the presence of repeat regions and consequently spurious overlaps in the vicinity of a BAC complicates efforts to correctly associate the relative position of a candidate mate pair with respect to the BAC. We therefore impose several constraints on a candidate mate pair if it is to be validated by a BAC. They are as follows:

- **Chain length constraints.** We require that the calculated distances of mate pairs from the BAC ends are consistent with the estimated BAC lengths. Such a constraint is necessary in dealing with regions that are (exactly or approximately) repeated within the span of a given BAC; these kinds of repeats can cause discontinuous jumps over sections of the genome, leading to chains that arrive at the other end of the BAC prematurely. Unfortunately, even a chain composed of only non-spurious overlaps will have some uncertainty in its length due to the variation of insert sizes within a library. This, coupled with the uncertainty in the length of the BAC insert makes it necessary to accept mate pairs whose distances from the BAC ends falls within a certain range. We note that paths that encounter short repeats (relative to the insert sizes making up a chain) that are repeated outside of the span of a BAC will not in general, cause a problem since it is unlikely that they will contribute mate pairs that will be encountered by the wavefronts originating from both ends

of the BAC. In our experiments on the faux Celegans data, we have accepted chains that have calculated lengths that are within 30% of the mean BAC length. Simulations have shown that a larger range is preferable to a narrow one.

- **Directional constraints.** A further requirement we have imposed is that an insert should be used from both of its ends (reads). This constraint has been necessary to counter the effects of "reverse" repeat regions where a section of DNA is repeated (exactly or approximately) in reverse order somewhere else in the genome. These types of repeats cause paths to reverse direction and fold onto themselves.

- **Recording of distances.** As we follow the mate pairs that are used in extending a wavefront from a BAC end, we allow for the possibility of an insert being used multiple times. This is especially necessary when the variation of the insert sizes within a library is large or when a combination of libraries is used in forming chains as in a chain of alternating 10k's and 2k's. However, only the first calculated distance is used.

The procedure discussed above is not limited to any particular set of insert libraries. For specificity, we might assume we had three libraries of inserts of mean length 2 Kbp, 10 Kbp, and 150 Kbp. We refer to the individual inserts as a **2K, 10K**, or a **BAC**, respectively. Of course only the sequences of the end reads of these are available. For the results mentioned in this paper, we have spanned (i) BAC's with 10K's, (ii) BAC's with 10K's but where alternating 2K's are allowed and (iii)

10K's with 2K's.

2.4.3 Chain assembly

The next step is to start assembling the genome for the neighborhood-based chains. The important point is as follows. Since we have a bit of a global structure, we can now create an assembly in which we only use an end of an insert if we can also use the other end in a position appropriate for the length of the insert. We will describe this procedure in more detail after describing our overlap procedure. Our overlap procedure is important because it produces relatively few spurious overlaps. For the shooting method to work, we cannot have the chains starting from one read quickly reaching everywhere in the genome, as would be the case if each read had on the average 10 spurious overlaps for each true overlap.

The goal of this method is to build a chain through the entire sequence of a chromosome using only the validated mate pairs. Once the chain is obtained, the sequence of BACs is found as the BACs that validate the mate pairs along the chain and this sequence is our first guess for virtual physical map. We proceed as follows.

Step 1. For each BAC B all mate pairs that belong to all possible valid chains through B are found and marked as validated by B.

Step 2. Starting from a random mate pair P_0 that is validated by at least a number of BACs equal to a half of BAC coverage, a chain is built according to the following scheme. We record the BACs that validate P_0 and call it the **current neighborhood** $\{B\}$. Then we pick the set of validated mate pairs $\{P\}_1$, that

overlap one end of P_0 in the relative reverse direction. Next, we assign a score to the members of $\{P\}_1$ as the ratio of the weighted number of validating BACs, that are also in $\{B\}$, to the total number of validating BACs for each mate pair in $\{P\}_1$. For each BAC we record how long it has been in current neighborhood (erroneously placed BACs can be found later using the information about how long the BACs were present in the current neighborhood, and this is how we refine our first guess for virtual physical map). In calculating the scores for candidate mated pairs one of which will become the next link on the chain, the BACs that have been on the current neighborhood longer are considered more reliable, and they are accordingly given a larger weight. This way we ensure the slowest rate of neighborhood change. We pick the mate pair P_1 with the highest score and add its validating BACs into the current neighborhood $\{B\}$, noting their relative starting position with respect to the BAC list that we have started with. In general BACs are added to the current neighborhood and removed from it as they are no longer validating recent mate pair additions to the chain. The process of building the chain is carried on until there is no way to continue with a validated mate pair or if all mate pairs in the current set $\{P\}_k$ have a score of zero. If one of these situations occurs, then we use a standard tree search algorithm that allows us to step back for a certain number of steps and attempt to find an alternate chain. If it proves impossible to extend the chain any further, we stop and go back to P_0 and use the same technique to extend the chain in the opposite direction.

Once a chain is found, we verify it for consistency by starting at each end and using our software to build alternate chains. Our algorithm is stable enough to

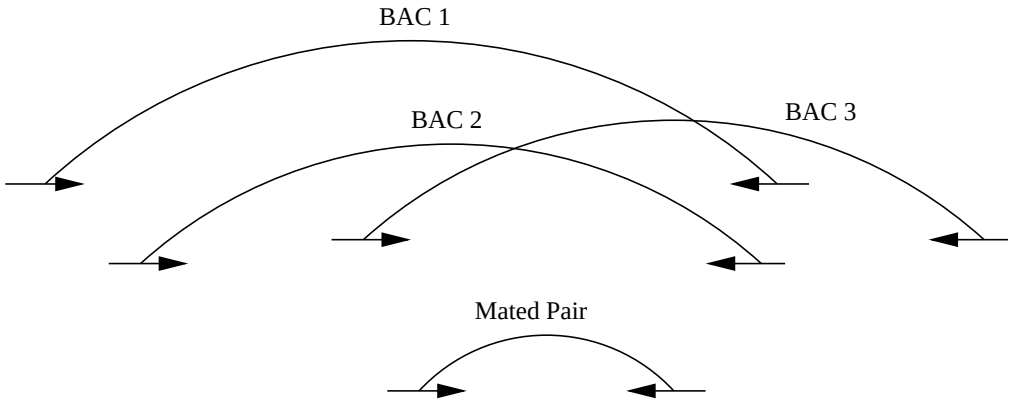

Figure 2.3: A small insert can be a part of chains spanning several large inserts, shown here as BAC 1, BAC 2 , and BAC 3 (of all the inserts making up the chains across the BACs, only one that is common to all three is shown). We say that the small insert has "neighborhood" of BAC 1, BAC 2 , and BAC 3.

construct a chain that is almost identical when going in the opposite direction. We then compare the physical maps belonging to these alternate chains and if inconsistencies are found, we split the chain into pieces. If the resulting chain is consistent in both directions, we call it a **reliable chain**. Once we have a set of reliable chains covering the genome multiple times, we then use the virtual physical map information to piece them together into one reliable chain spanning the entire contiguous sequence.

The advantages of our technique are: (i) it allows one to build a physical map and identify the relative positions of the mate pairs and BACs in the genome at low coverage levels, and thus at relatively low cost; (ii) it saves money on doing the fingerprinting, providing the same results in the end; (iii) it is easy to implement and fast to execute.

In Stage 2 of our assembler we use only the mate pair data to find the relative positions of BACs, or, in other words, build a virtual physical map of the genome. We also produce a reliable chain of mate pairs spanning the entire sequence of the chromosome. We are able to create a physical map and a chain path continuously spanning over 99% of the *C. elegans* genome in our experiments.

If we are provided with enough BACs so that their end reads cover the genome 0.05 times on the average, then as one moves along the genome, the left end of a BAC will be encountered on the average every 20,000 bases. Since BACs are typically 150Kbp long, it follows that the average 10K will lie in about 7 BACs. A 10K will typically belong to chain spanning many of these. We say that the BAC **validates** the 10K. The **neighborhood of a 10K** is the set of all the BACs which validate the 10K. The next step is to create a completely new chain of 10Ks. (Some 2Ks may be used sparingly). In topological terminology, a neighborhood N of a set P contains points that close to P but is also permitted to contain some points that are far away. Due to repeats, the neighborhood of a 10K may indeed contain BACs that are far from the 10K. The goal is to create a chain as long as possible, perhaps running the length of a chromosome, such that the neighborhoods change slowly along the chain. We create a chain of validated 10Ks with the property that as one moves along the chain, the neighborhoods changes as slowly as possible. The following small-scale example may help. Assume that P_1, P_2, P_3, P_4, P_4' are validated 10K's and that the following table lists their neighborhoods (reduced in number here for simplicity). The BACs are labeled B_1, B_2, etc.

P_1 is validated by B_1

P_2 is validated by B_1 and B_2

P_3 is validated by B_1, B_2, B_3, B_4

P_4 is validated by B_1, B_2, B_5

P_4' is validated by B_3 and B_4

Assume further that P_1, P_2, P_3, P_4 is a chain as is P_1, P_2, P_3, P_4'. The idea here is that P_3 is in a repeat region and that while B_1 is close to B_2 and B_3 to B_4, the first pair may be far from the second pair. Should we choose the chain ending in P_4 or P_4'? The answer is P_4, since its BACs have been in the chain longer than those of P_4'. Choosing P_4 results in a chain in which the neighborhoods change more slowly. BACs that have been in the chain for fewer neighborhood 10Ks are weighted less in this consideration.

Such a **neighborhood-based chain**, allows us to see the overall structure of a large piece of the genome. The BACs like B_3 and B_4 that are in neighborhoods of the chain only briefly, that is for say one or two 10Ks are ignored. Then we have the beginnings of a global map for this region with the chain running from B_1 into B_2 and then possibly into B_5. Of course this global map can and will be significantly refined.

2.4.4 Chain assembly results

To test our methods, we used the *C. elegans* faux reads described in Section 3.4. We validated 2K's and 10K's against BACs and 50K's. We have also validated

the 2K's against 10K's. After validating the 2K's and 10K's, we end up with 98% of them being validated by at least one BAC, 50K or 10K. Then we built chains with 2K's and 10K's. Fig. 2 shows chains obtained, that span the chromosomes 3 and 4. The individual mate pairs are shown as dots. The vertical coordinates of the dots show the relative locations of the mate pairs within chains, and the horizontal coordinates the their actual locations in the genome, so the correct chain should look like a continuous upward – sloping line. The chain through Chromosome 3 contains two misplaced mate pairs. We have examined this case and came to the conclusion that in terms of consensus sequence it would not constitute a problem, because these mate pairs are completely inside identical copies of repeat regions and the algorithm just picked up the copies of mate pair belonging to different copy of the repeat region.

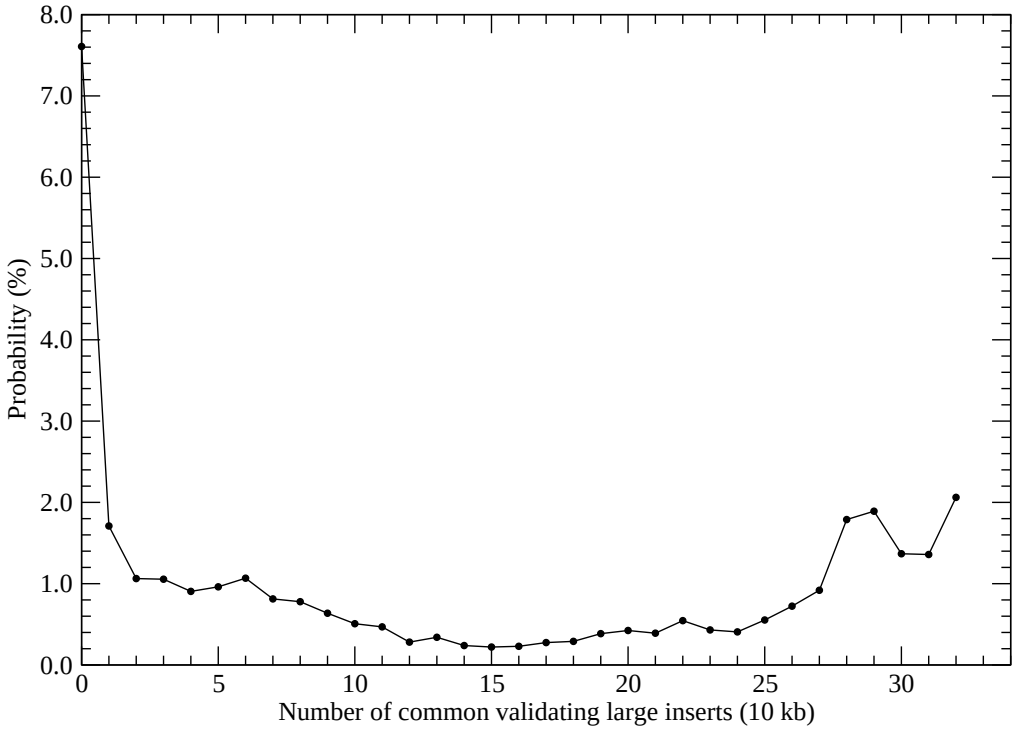

Figure 2.4: The probability of encountering a spurious reverse overlap between reads belonging to 2kb inserts after they have been validated using 10kb inserts. The horizontal axis denotes the number of common validating 10kb inserts the 2kb inserts have. The peak to the left of the figure is due to two effects: i) Most (reverse) spurious overlaps between 2kb inserts do not have common validating 10kb inserts, and ii) a significant number of small inserts near the ends of the big inserts are not validated. In other words, during validation, it takes some steps before a "wavefront" of small inserts starting from one end of a large insert to "decohere" due to insert size uncertainties. We note that the ratio of false (spurious) overlaps to true overlaps in this dataset is about 1 %.

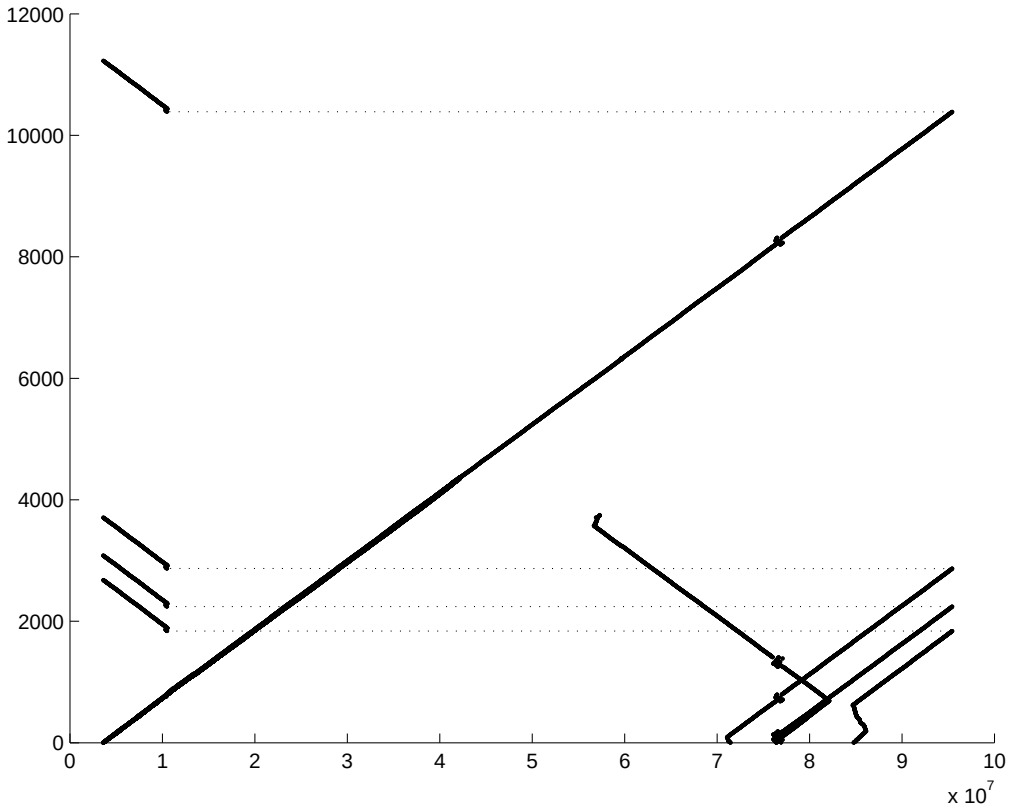

Figure 2.5: The chains through the entire genome of the *C. elegans*. The individual mate pairs are shown as dots. We number the mate pairs consecutively according to their position on the chain. For each mate pair the vertical coordinate is its number in the chain and the horizontal coordinate is its actual position in the genome in units of bases.

Appendix A

Minimizers

Here we describe a method that we use to greatly reduce the amount of time and storage needed for overlap pairing. We select an ordering for the set of all k-mers of fixed length k. To facilitate this ordering, we assign a different digit $(0, 1, 2, 3)$ to each of the 4 bases A, C, G, T and then, thinking of k-mers as k-digit numbers, use the numerical ordering.

To find a "minimizer", we examine m consecutive k-mers and select the smallest, in the sense of our chosen ordering. More precisely, if S is a string of $m + k - 1$ bases, then it contains exactly m consecutive k-mers, where "consecutive" means each k-mer is shifted by one base from the previous one. (The "first" k-mer of S begins with the first base, and the jth k-mer begins with the jth base.) If M is the smallest k-mer in S, then we say M is the **simple minimizer** for S. ("Simple" denotes the fact that we do not examine the k-mers of the reverse complement of S.)

For a string S or a read X, we denote its reverse complement by $-S$ or $-X$. An **(absolute) minimizer** for S is a k-mer that is the smallest of all the k-mers in both S and $-S$. By definition, S and $-S$ have the same minimizer.

Finally, we define the **m-minimizers** of a read X to be those k-mers in X that are absolute minimizers for some substring S of X with length $m + k - 1$. In

nontechnical discussion we will often call a/ m-minimizers simply a **minimizer**.

Fact: If two reads have an error-free overlap of at least $m + k - 1$ bases, then they must have a common m-minimizer.

Fact: A read has the same m-minimizers as its reverse complement.

We say the **offset of a k-mer** M in a read is n if the first base of M is the n-th base of the read.

Fact: Two consecutive m-minimizers in a read have offsets differing by at most m.

Hence if $m \leq k$, then Property M1 above holds. That is, the collection of m-minimizers cover the entire read, except for at most $m - 1$ bases at each end. Property M2 above also holds, in that two reads with error-free overlap of many more than $m + k - 1$ bases will have several common m-minimizers.

Example. Let $k = 20$ and $m = 20$, and let X be a read of length 400. Then it must have at least 19 minimizers. The first minimizer has offset at most 20, the second at most 40, etc. Similarly, if two reads overlap by 400 bases then they must have at least 19 common minimizers.

There are a number of orderings that can be used for defining the minimizers of a read; below is one example. What is important is that the same ordering should be used for all of the reads.

Example. For the results reported in this paper, we assign the values $0, 1, 2, 3$ to C, A, T, G respectively for the odd numbered bases of m-mers and assign $0, 1, 2, 3$ to G, T, A, C respectively for the even numbered bases. The purpose is to avoid having minimizers that start with relatively error-prone strings like $AAAAAAAA$

or *CCCCCCCC*.

Appendix B

UMD Assembly Pipeline

In this section we present the UMD assembly pipeline. We start with a set of error-corrected and retrimmed WGS reads and BAC reads along with their overlaps obtained from UMD overlapper[32]. The general strategy is to pick a reliable subset of the total set of all overlaps such that the resulting assembly had the most sequence that matches the finished sequence with the least error rate. We provide details on how we compare the assembly to the finished sequence in the Appendix. We use a version of *Phrap* [17] contig assembly software that we have modified to ignore overlaps that are not contained in a list that we provide. While Phrap computes its own overlaps, our modified version excludes overlaps that are not on our list, forcing Phrap to use overlaps from the intersection of two sets: the overlaps it computes, and those that we provide. We call this modified version of Phrap, "PhrapUMD". The technical details are presented in the Appendix C. Finally, we use ATLAS to build scaffolds from reads and contigs built by our version of Phrap.

B.1 UMD Unitigger

Our first take on creating a reliable set of overlaps was to design a unitigger and declare all overlaps used in the unitigs to be reliable. On the scale that trades erroneous connections for unitig length, our experience is that existing unitiggers do

a reasonable job of avoiding erroneous connections, but sometimes extend unitigs at the expense of an occasional erroneous connection. The current version of the UMD unitigger, in contrast, is designed to lie far towards the conservative end of this scale, with the goal of making absolutely no incorrect connections. Here, "no incorrect connections" means:

(a) If we are in a non-repetitive part of the genome, the unitig will contain all reads that come from this part of the genome (unless they are very short or have excessive errors) and no others.

(b) If a unitig represents more than one part of the genome, all reads that come from these parts of the genome are in the unitig and no others.

For example, if a unitig of length 5,000 bases represents portions of the genome beginning at bases 0, 100,000, and 200,000, and no others, then the unitig will have all reads that lie between bases 0–5,000, 100,000–105,000, and 200,000–205,000, and not contain any others, with the same caveat about length as in case (a) above.

The approach to the prior parts of our assembler (overlapper and error corrector) has been to avoid making mistakes, at the cost of missing some opportunities to improve various statistics. The goal of this unitigger is the same. The UMD unitigger typically yields shorter unitigs than other unitiggers, but our experience is that the probability of a misassembly in our unitigs is smaller than in currently available unitiggers. We tested the UMD unitigger on faux shotgun data from both the 1.64 megabase genome of the bacteria *Campylobacter jejuni*, and the 100 megabase genome of the nematode *C. elegans*. No misassemblies were found, where a misas-

NHGRI	GDOJ	GEFH	GMEZ	GSGV	GXWB	GYMN	GZLE
BAYLOR	GBYZ	GDWN	GEXM	GGKP	GIXT	GQQD	GRMX
	GSFK	GSTA	GTGF	GXFC	GZJC	KBQM	KDFE

Figure B.1: The 21 Baylor BACs studied, 7 covered by sequence independently finished at NHGRI, and 14 finished by Baylor.

sembly is defined as in (a) and (b) above. The bacteria had 15X coverage, while *C. elegans* had 6X coverage. Our hope in taking such a conservative approach is that if a mistake is made later in assembly, one can eliminate the places one needs to look for errors since they don't occur in the unitigs.

B.2 Binning of reads

We start with about 33 million WGS rat reads, along with some BAC reads (reads that are known to belong to the interior of certain BACs). The first step of our procedure on the rat is to run the UMD overlapper [32] on the entire set of rat reads (both BAC and WGS reads taken together as one homogeneous set), which have been trimmed to the traditional 2% error rate. This gives us about 200×10^6 overlap pairs, with the reads having been error-corrected out to their 2% trimmed length. Then we use our unitigger to build unitigs (uniquely assemble-able sequences). We found that overlaps of reads in a unitig are extremely reliable. The following reads are associated with a BAC B: the set B_0, which is the set of BAC reads for B; the set A_1, which includes any read that directly overlaps a read in B_0 (including BAC reads from other BACs); the mates of those reads in A_1; and reads

sharing a unitig with a read from B_0, as long as the read is within 2000 bases of the nearest B_0 read in the unitig. This last condition is designed to exclude reads that may lie beyond the end of the BAC, since even with 1X coverage by BAC reads it would be very unlikely to see 2000 bases inside a BAC without seeing a BAC read. We tested the efficacy of this approach by looking to see how often both mates of a 2kbp-long WGS insert were correctly binned in this fashion. In a typical BAC with about 2,000 WGS reads, only 5–10 binned reads had mates that were interior to the BAC (according to comparisons with finished sequence) but not binned. The remaining mates were exterior to the BAC, and so it was correct not to include them in the bin. A WGS read is allowed to be binned into more than one BAC, for two reasons: first, BACs overlap and so a WGS read may lie in the area of intersection and legitimately lie in both BACs; and second, because we don't want to allow an incorrect binning of a WGS read to exclude that WGS read from its correct bin. Reads that globally appear (illegally) in multiple places in the assembly are resolved later.

B.3 Contig merging

The results of assembly using reliable overlaps have shown that by imposing a restricted set of overlaps onto PhrapUMD, we are able to build a set of high-quality contigs. However, our conservative approach of handling repeats can result in some ambiguous repeat overlaps not being resolved by overlap information alone.

One way around this problem, which we have adopted, is to consider adjacent

contigs in the scaffold and determine which of these can be merged without the introduction of additional reads. In this procedure, PhrapUMD is run once on the full set of BAC reads with the conservative set of overlaps as a constraint. The resulting contigs are then processed using ATLAS-scaffolder. We then consider the reads flanking the gaps in the scaffold and see whether they overlap according to the less stringent set of overlaps. If this is the case, the overlap between these two reads is added to the original set of conservative overlaps. We find that a second pass of PhrapUMD using this extended set results in many fewer contigs without sacrificing the error rate of the resulting sequence or the fraction of finished sequence spanned. These contigs are then re-scaffolded to get the final result. We call this procedure "contig merging". In this way, we effectively force Phrap to use mate pair information in building contigs.

In practice, it is found that adding *only* the overlap between reads on either side of a gap is usually not enough; for consistency in the contig layout, other overlaps may need to be added. It is also possible that adjacent contigs will not merge because either or both of the contig ends are spurious. This case is checked by alternately removing the last reads from the adjacent contig ends and trying a merge on the resulting stripped contigs. Furthermore, an inspection of the scaffolds produced by ATLAS has shown that small contigs can be erroneously placed behind others. We have successfully corrected may of these cases by considering the merging of small contigs not only with their given neighbors but also with their neighbors

one contig away.

Appendix C

Comparing scaffolded assemblies

C.1 Scaffolds and Xcontigs

A scaffold consists of ordered and oriented contigs. The unknown sequence between a pair of contigs is represented by a string of 'N's whose length is approximated from mate pair information. Since these bases are unknown, it is unclear how we should evaluate them in an alignment. We believe that they should count neither as errors, nor as contributing to "total sequence", and we prefer to avoid their evaluation altogether. To that end, we introduce the idea of an "Xcontig", which is a maximal contiguous sequence of non-'N' bases in a scaffold — that is, a maximal length sequence in a scaffold containing only the letters `A, C, G,` and `T`. Some Xcontigs can be very short. In the following, we consider only Xcontigs that are 1kbp or longer. We then match these Xcontigs against finished sequence.

For a given BAC, we count the number of bases of finished sequence matched by Xcontigs longer than 1kbp. (That is, if more than one Xcontig covers a certain finished base, the base is only counted once.) The aggregate results stated below are simple sums over all BACS, as if all BACs were independent (despite their sometimes overlapping).

C.2 Using Blastz to match Xcontigs to finished sequence

We use Blastz (**citation**) to do the matching. Experience has shown that if an Xcontig has more than one Blastz match to finished sequence, the longer match is not necessarily the more desirable one. Often, another match with a slightly shorter length but many fewer errors will be present, and better alignments can be found by defining a score S that severely penalizes errors. If K is the factor by which we penalize each base error, we define the score of an alignment to be

$$S = \text{alignment length} - K * (\text{number of discrepancies}).$$

for each alignment. If an Xcontig has an alignment of at least 1kb with a positive score then the highest-scoring such alignment is called the successful match. We have chosen $K = 125$, which means that a successful match can have at most a 0.8% error rate, compared to finished sequence. The "tails" of an Xcontig are the parts at either or both ends that are outside the successful match. See Figure C.1. Tails delimit a match but are not involved in its scoring. Xcontig sequence that runs off the end of finished sequence is not considered a tail and is not counted anywhere. This is justified because the precise endpoint of finished sequence in a BAC is arbitrary, and although a correct scaffold can extend beyond the boundary of this finished sequence, we restrict our analysis to Xcontig sequence that matches finished sequence.

We have seen a number of cases where Blastz fails to identify ends of alignments as tails. These ends can involve large numbers of indels or substitutions (eg. up to 50% of the bases) within a very short span (typically tens of bases) in what

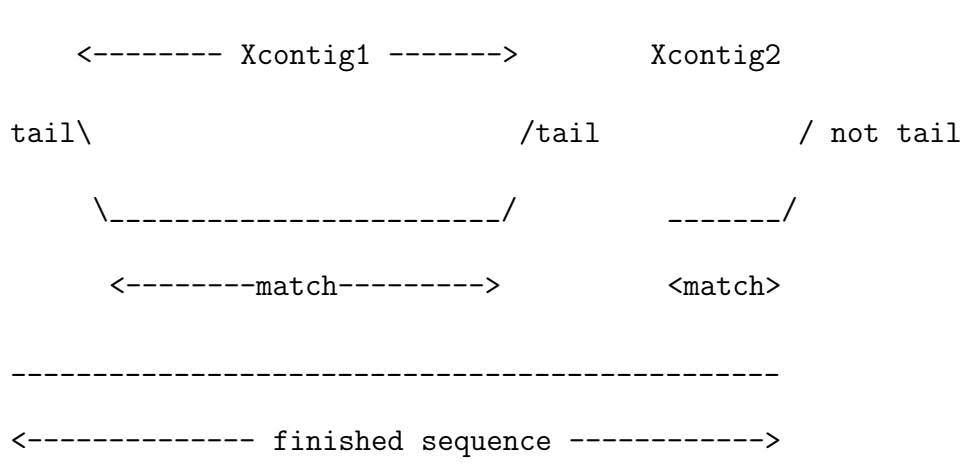

Figure C.1: Schematic diagram of two Xcontigs that match finished sequence. Xcontig1 has a long match and short tails on both ends that do not match the finished sequence. Xcontig2 has no tail, because on the left it matches finished sequence all the way to its end, and on the right it runs off the end of finished sequence and is thus unverifiable. The latter "non-tail" sequence is not counted anywhere in our analysis.

is otherwise a very solid match. Such cases should be part of a tail, but are not currently counted as such, so we call them "tail errors". We do not currently account for them, and so they artificially inflate the error rate.

The default gap penalty for Blastz is set too low for our taste. A common case involves the deletion of a number of consecutive bases followed immediately by the insertion of a roughly equal number of bases ("gap-penalty errors"). Such an alignment can be expressed more concisely in terms of approximately half the above number errors as substitutions. This results in an error rate which can be as much as twice as high as it should be. The parameters we have used for Blastz comparisons are `C=2 W=16 T=0 K=25000`, where `K` is a threshold that helps eliminate spurious matches (the default value for which is `K=2500`), `W` is the word length used in

initiating a match, and `C=2` ensures that blastz uses a "chain and extend" approach in matching sequences (the default is to not chain).

C.3 Defining errors

We define the total error rate across a set of BACs to be

$$\frac{\sum [\text{insertions, deletions, and substitutions inside a match}]}{\sum [\text{matching length of Xcontig}]},$$

where the sums are carried out over all matching Xcontigs in all BACs. (Note that the denominator here is the sum of the matching lengths of Xcontigs rather than the number of finished bases covered, and that all errors in the Xcontigs are counted. For example, if two Xcontigs cover a given finished base, and both get the base wrong, then both errors are counted.)

This error rate will include "base errors" which are isolated errors scattered throughout the assembly, "tail errors" as defined above, and "internal errors" which show up as contiguous blocks of insertions and deletions far from the edges of the matching region (if these occur near the ends of the match, we would like to count them as tails, but tail errors frequently confound this). Ideally, internal errors should be distinguished from base errors but we do not do this here.

The largest internal error we have observed is a discrepancy concerning the number of occurrences of a 19-base tandem repeat in the BAC "GSTA". In this case, nine reads overlap the region. Retrieving the uncorrected version of these 9 reads, visual inspection indicates that all 9 reads agree with our assembly and none agree with the finished sequence. This suggests that the finished sequence contained

3 extra copies of this 19-base repeat element, for a total of 57 insertion errors in the finished sequence. It is possible that the difference arises from a polymorphism between the individuals donating the finished vs. the unfinished sequence, but in any case we have modified our version of GSTA's finished sequence to conform to our version of the assembly since it is clear our assembly is correct given the reads we use.

We also note that discrepancies between an assembly and its finished sequence can arise at bases where the finished sequence (and possibly also the corresponding base in the Xcontig) itself is of low quality, that is, with quality value 20 or less. We refer to this situation as "sequence ambiguity".

In our calculations of error rates, we make no distinction between the above kinds of errors. While those such as internal errors may be legitimate sources of discrepancies, others such as tail errors, sequence ambiguities and gap-penalty errors will artificially drive up error rates reported in calculations. As the quality of assembled sequence improves and we reach discrepancy rates between assembly and finished sequence of less than 1 in 10,000 (as we do below) we can expect this issue to become more relevant.

Appendix D

PhrapUMD

The aim of this section is to give a bief overview of the workings of Phrap for the purpose of identifying the location and context of the UMD changes. Phrap

is a program that will take read sequences (and, optionally, quality values) and output contig sequences and associated files. It follows a set of stages that can be summarized as "overlap", "layout", and "consensus". The "overlap" stage is responsible for establishing a database of read pairs that plausibly overlap. A "seed-and-extend" procedure is used here, where the read sequences are first scanned and a list is made of all the possible k-mers that occur in all reads. Pairs of reads that have common k-mers are identified as "candidate" overlaps and a plausible subset of overlaps are then subjected to a Smith-Waterman analysis.

The crucial UMD modification involves intercepting function calls to make_new_cand_pair() and creating such a pair only if this overlap exists in the approved overlaps database. We reproduce below the relevant part of the Phrap code (for context) with our single-line modification. For brevity, we present only changes to the code that alter the logic of Phrap; any additional supporting subroutines are not shown.

```
void make_new_cand_pair(entry1, entry2, start1, start2, reverse)
     int entry1, entry2, start1, start2;
     int reverse;
     /* offset currently not used -- but should be! */
{
  Cand_pair *pair;
  Cand_pair *get_cand_pairs();
  int off, off_min, off_max, temp;
```

```
  Segment *insert_segment();
  Seq_entry *get_seq_entry();


  /* ------------- THIS LINE INSERTED BY UMD --------------- */
  if(!exists_in_approved_db(entry1, entry2)) return;


    ...


}
```

[1] F. R. Blattner et al. "The Complete Genome Sequence of E. Coli", Science 277, 1453–1474 (1997).

[2] H. W. Mewes et al., "Overview of the yeast genome", Nature 387, 737 (1997).

[3] The Genome Sequencing Consortium, "Genome Sequence of the Nematode *C. elegans* : A Platform for Investigating Biology", Science 282, 2012–2021 (1998).

[4] S. Batzoglou et al. "ARACHNE: A Whole Genome Shotgun Assembler" Genome Research 12, 177-189 (2002).

[5] E. W. Myers et al., "A Whole-Genome Assembly of *Drosophila*", Science 287, 2196–2204 (2000).

[6] A. Coulson et al., "Genome linking with yeast artificial chromosomes", Nature 335, 184 (1988).

[7] J. Sulston et al., "The *C. elegans* genome sequencing project: A beginning" Nature 356, 37 (1992).

[8] R. Wilson et al., "2.2 Mb of contiguous nucleotide sequence from chromosome III of C. elegans" Nature 368, 32 (1994).

[9] J. C. Venter et al., "The Sequence of the Human Genome", Science 291, 1304–1351 (2001).

[10] J. Kececioglu and J. Yu, "Separating repeats in DNA sequence assembly", in *Proceedings of the 5th ACM Conference on Computational Molecular Biology*, ACM Press, 176–183 (2001).

[11] M. T. Tammi, "Software Tools and Algorithms for Shotgun Sequence Assembly", Ph.D. Dissertation 2002, Acta Universitatis Upsaliensis, Uppsala (2002).

[12] P. Green, `http://phrap.org/phrap.docs/phrap.html`

[13] M. Waterman, *An Introduction to Computational Biology*, Chapman and Hall (1995).

[14] P. A. Pevzner, H. Tang, M. S. Waterman, "A new approach to fragment assembly in DNA sequencing" in *Proceedings of the 5th ACM Conference on Computational Molecular Biology*, ACM Press, 256–267 (2001); "An Eulerian path approach to DNA fragment assembly", Proc. Nat. Acad. Sci. 98, 9748–9753 (2001).

[15] J. K. Bonfield, K. Smith, R. Staden "A new DNA sequence assembly program", Nucl. Acid Res. 24, 4992–4999 (1995).

[16] R. D. Fleischman et al., "Whole-Genome Random Sequencing and Assembly of Haemophilus influenzae Rd.", Science 269, 496–512 (1995).

[17] Green, P., PHRAP documentation (1996).

[18] P. Green, "Against a whole-genome shotgun", Genome Res. 7, 410–417 (1997).

[19] X. Huang, "A Contig Assembly Program Based on Sensitive Detection of Fragment Overlaps", Genomics 14, 18–25 (1992).

[20] X. Huang, "An Improved Sequence Assembly Program", Genomics 33, 21–31 (1996)

[21] R. M. Idury and M. S. Waterman, "A New Algorithm for DNA Sequence Assembly", J. of Comp. Bio. 2(2), 291–306 (1995).

[22] J. D. Kececioglu and E. W. Myers, "Combinatorial Algorithms for DNA Sequence Assembly", Algorithmica 13, 7–51 (1995).

[23] S. Kim and A. M. Segre, "AMASS: A Structured Pattern Matching Approach to Shotgun Sequence Assembly" J. Comput. Biol. 6(2), 163–186 (1999).

[24] J. Mullikin et al., `ftp://ftp.ensembl.org/traces/human/qual/Chr_20/`.

[25] E. W. Myers, "Toward Simplifying and Accurately Formulating Fragment Assembly", J. Comput. Biol. 2, 275–290 (1995).

[26] F. Sanger, S. Nicklen, A. R. Coulson, "DNA sequencing with chain-terminating inhibitors", Proc. Natl. Acad. Sci. U.S.A. 74, 5463–5467 (1977).

[27] G. G. Sutton et al., "TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects", Genome Sci. and Technology 1, 9–19 (1995).

[28] J. C. Venter et al., "Shotgun Sequencing of the Human Genome", Science 280, 1540-1542 (1998).

[29] G. Vogel, "Sanger Will Sequence Zebrafish Genome", Science 290, 1671 (2000).

[30] J. L. Weber and E. W. Myers, "Human Whole-Genome Shotgun Sequencing", Genome Res. 7, 401–409 (1997).

[31] Dan Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge Press (1998).

[32] M. Roberts et al., "Reducing storage requirements for biological sequence comparison", Bioinformatics 20, 18 (2004).