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

Cevat Üstün, Wayne Hayes, Brian Hunt, Michael Roberts, James Yorke, Aleksey V. Zimin [*]
Institute for Physical Science and Technology
University of Maryland
College Park, MD 20742-2431
and
Paul Havlak
Baylor College of Medicine

DRAFT of March 10, 2005

Abstract

We present an algorithm to generate a set of highly reliable overlaps based on identifying repeat k-mers. This method does not require uniform coverage. We also describe an error correction method based on multi-sequence comparison, that allows extending usable sequence in reads trimmed to 16% error rate. We use a version of the *Phrap* assembly program that uses only overlaps computed by the UMD overlapper, called *PhrapUMD*. We integrate the University of Maryland 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 up to 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 93.7% to 96.7%, while simultaneously reducing the base error rate from 4.5 to 2.0 errors per 10,000 bases.

[*]For questions or comments, please write to `alekseyz@ipst.umd.edu` and `yorke@ipst.umd.edu`.

1 Introduction

Improving WGS. Whole genome shotgun (WGS) assembly has created draft versions of several large genomes, including the fly *D. melanogaster*, rat *Rattus norvegicus*, and human. Although the majority of the cost in using WGS is in producing the read sequences, the assembly is still an extremely important factor in producing the draft genome. Better assembly methods better utilize the expensive read data, reducing the amount of money and effort needed for finishing the genome. The Baylor College of Medicine (Baylor) assembler ATLAS uses 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 [1] [2].

Evaluation and results. The sequencing for the rat genome project was done using a mix og WGS data and BAC read data, so that each BAC could be assembled individually. 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. We used the same data that was used in ATLAS assembly. Based on a comparison with the assemblies of 21 finished BACs, using our extended reads and reliable overlaps (to be discussed below), we recover 3% more sequence matching to the finished sequence at less than half the overall error rate, thus reducing the effort requred to finish our draft.

ATLAS. To give the flavor of the ATLAS strategy for rat assembly, we summarize its procedures as follows. See [1] 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 that BAC. ATLAS then applied Phrap [3] to the binned reads to produce an assembly of the BAC. Then ATLAS created a draft 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.

1 The UMD overlapper [4] corrects errors in reads and produces overlaps with a goal of missing at most one true overlap in 10^8 (when reads overlap by at least 40 bases).

2 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 the expected error rate reached 8% to 16%.

3 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 be 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, according to the mate pair information.

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 standard Phrap meant that UMD high-quality overlap information was used 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 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 if it 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 [4] creates a high quality 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 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 of all seeds that belong to repeat regions. A 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 the PhrapUMD design in the appendices A-D.

2 Methods

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 [5]. 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 from 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 have an exact overlap overlap of at least 40 bases, then 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 is independent of the coverage, and that could be applied to any data set without modifications.

Our method is the following. We create a list of all minimizers encountered in all reads, sorted by minimizer value. We then examine the minimizers and if a minimizer is found to be common to a pair of reads that eventually are reported as non-overlapping, we declare that minimizer *repeat* or *unreliable* minimizer. 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. The latter requirement is imposed to avoid declaring as reliable repeat-induced overlaps that have errors in the read sequences. This method produces the results described in Section 4.

2.2 Read Extension and error correction

Traditionally, excessive sequencing errors have tended to confound assembly. Thus reads are usually 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 [4] 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.

```
Case 2 is:
Satisfied              Not Satisfied
----A----                ----A----
----A----                ----A----
----A----                ----A----
----T----                ----G----
----G----                ----G----
```

Figure 1: 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.

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 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.

Obviously, we do not want to use spurious overlaps to compute corrections. We assume that the number of differences per overlap is distributed according to Poisson distribution and we use this to estimate the probability that the overlap is valid. We only reject an overlap if the probability that it is valid is less than 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. We consider this to be an extremely conservative method of error correction.

2.2.1 Retrimming excessive errors off the ends of extended reads

To recap, we start with reads that are trimmed (before correction) to a rate of X%, and then run our overlapper and error-correction routines, as described in [4], 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

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

#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 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.

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 presents the average read lengths obtained as a function of trimming, with and without retrimming, across the 21 BACs listed in Figure 3. 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 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 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.

%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	5440	18930	4247332	4322053	40661	95.805	1.849	558
08	6078	16871*	4262835	4335172	37130	96.155	1.552	481
10	5123	23476	4271302	4354607	47885	96.346	1.529*	480
12	5111	18986	4285519*	4377077*	54808	96.666*	1.889	491
16	4998*	17382	4227365	4322722	61108	95.355	2.401	525
ATLAS	14920	74010	4140003	4189363	9980*	93.384	4.473	377*

Table 3: Assembly results obtained using only reliable overlaps. Numbers that are the best in the 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	7854	17526	4249970	4287896	3880	95.865	1.670	450
08	6317	16871	4260174	4298241	2939	96.095	1.435*	388
10	6267	14646	4270713	4311479	5463	96.332	1.475	370
12	2896	10148*	4286188*	4327739*	4958	96.681*	2.010	363*
16	2878*	13066	4237589	4285272	1191*	95.585	1.928	420
ATLAS	14920	74010	4140003	4189363	9980	93.384	4.473	377

Table 4: UMD+ATLAS assembly results obtained using reliable overlaps and contig-merging overlaps compared with finished sequence. Numbers that are the best in the column are marked with a *.

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 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.

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

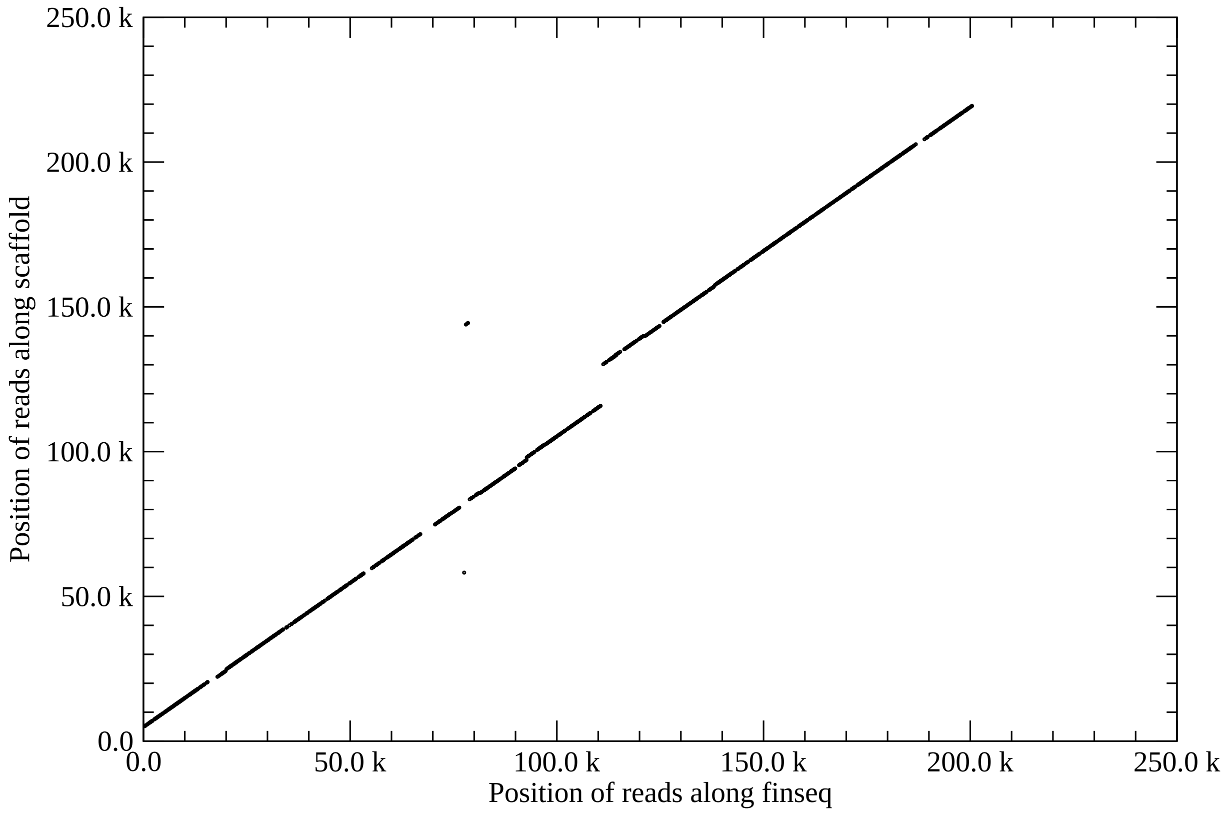

Figure 2: Our worst assembly across all 21 BACs. The individual reads are depicted as dots with their place in the scaffold as their vertical coordinate and their place in the finished sequence as horizontal coordinate. We used the best match according to blastz to place the reads.

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).

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 using Blastz software [6]. 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 Finished Sequence Matched Uniquely, 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 sequence 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 in matching Xcontigs that do not match finished sequence (substitutions, insertions, and deletions), divided by 10000. 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 scaffolds in the assembly of the 21 BACs.

The results for 21 BACs and for various read extension/retrimming tolerances are presented in Table 3. We used only the 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.

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 4 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 4 that using 10% or 12% extension is optimal for assembly purposes. We are now using the latest data in Freeze 04 to produce a complete draft Rat genome assembly with 12% extended reads. We intend to 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 our worst case for a particular BAC out of the 21 BACS that we examined. There are two reads on the ends of two contigs that are misplaced, possibly due to repeat regions, and one gap whose size is overestimated. Finishing our assembly may require much less effort and money, and the additional cost of creating our assembly is negligible.

4 Acknowledgments

This work was supported by NIH grant 1R01HG0294501, and NSF Grants DMS 0104087 and DMS 0312360.

Appendix A: 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 [4]. 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* [3] 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.

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 misassembly 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.

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

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

Figure 3: The 21 Baylor BACs studied, 7 covered by sequence independently finished at NHGRI, and 14 finished by Baylor.

UMD overlapper [4] 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.

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

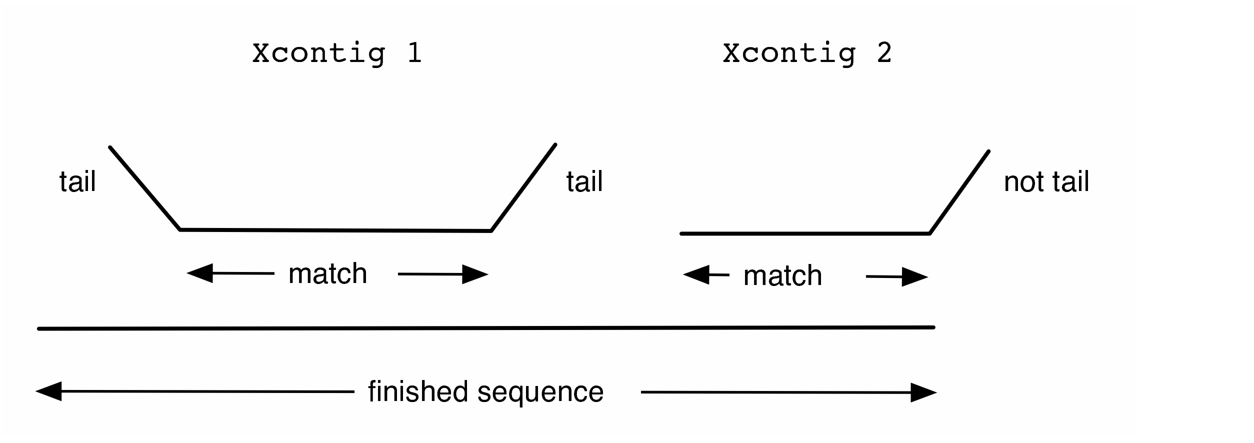

Figure 4: Schematic diagram of two Xcontigs that match finished sequence. Xcontig1 has a long match and short tails on both ends that do not match the finished sequence. Xcontig2 has no tail, because on the left it matches finished sequence all the way to its end, and on the right it runs off the end of finished sequence and is thus unverifiable. The latter "non-tail" sequence is not counted anywhere in our analysis.

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 B: Comparing scaffolded assemblies

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 4. 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 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` relates to the the gap penalty (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).

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 C: PhrapUMD

The aim of this section is to give a brief 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(pair->entry1, pair->entry2)) return;

    ...

}
```

References

[1] Paul Havlak, Rui Chen, K. James Durbin, Amy Egan, Yanru Ren, Xing-Zhi Song, George M. Weinstock, and Richard A. Gibbs. The Atlas Genome Assembly System. *Genome Res.*, 14:721–732, 2004.

[2] Rat Genome Sequencing Project Consortium. Genome sequence of the brown norway rat yields insights into mammalian evolution. *Nature*, 428:493–521, 2004.

[3] B. Ewing and P. Green. Phrap. Unpublished. `http://www.genome.washington.edu` or `http://www.phrap.org.`, 1994.

[4] Michael Roberts, Brian Hunt, James Yorke, Randall Bolanos, and Art Delcher. A preprocessor for shotgun assembly of large genomes. *Journal of Computational Biology*, 000:1–2, 2004. Accepted.

[5] Michael Roberts, Wayne Hayes, Brian Hunt, Steve Mount, and James Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18), 2004.

[6] Scott Schwartz, W. James Kent, Arian Smit, Zheng Zhang, Robert Baertsch, Ross C. Hardison, David Haussler, and Webb Miller. Human-mouse alignments with BLASTZ. *Genome Research*, 13:103–107, 2003.