Assembly Cookbook

Michael Schatz

Table of Contents

Borrelia afzelii (GBAF).....2
INITIAL ASSEMBLY.....2
UNITIG BLASTING.....3
MAIN CHROMOSOME IDENTIFICATION.....4
PLASMID ASSEMBLY.....7
APPENDIX.....8
Xanthomonas oryzae (Xoc).....11
Trichomonas vaginalis (TVG).....14
Drosophilia Virilis.....15
Brugia (BRG).....17
Gi PolyDNA Virus (GPV).....20
Cyanobacteria (GYMA & GYMB).....22

Borrelia afzelii (GBAF)

June 2005

INITIAL ASSEMBLY

The first step is to pull all of the non-trashed sequences (raw) from the database:

```
% mkdir /local/asmg/scratch/mschatz/GBAF/2005-6-21/pull
% cd /local/asmg/scratch/mschatz/GBAF/2005-6-21/pull
% pullfrag -D gbaf -raw -i -o gbaf
```

From prior experience, I knew that a .2 % unitig error rate would yield good results. I had determined this value by trying all values between 0 and 4% in .1% increments. I chose .2% because of the low number of correlated SNPs, but also having reasonable contig size. The error rate determines the maximum amount of difference the unitigger will tolerate for assembling reads together. Note this occurs after the kmer-based error correction from the overlapper, so only true differences should be left, i.e. differences in the repeat copies, and the unitigs won't be overly penalized from mere sequencing or basecalling errors.

I find it more convenient to use the runAmos launch script than the runCA script, so I use that instead (must be run on the opteron). The runca.amos.opteron script has all of the parameters used. It also points to a CA 3.06 which, incorporates a few changes made to the assembler since the last official release.

```
% mkdir /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/
% cd /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/
% ln –s ../pull/gbaf.frg
% runAmos –C runca.amos.opteron gbaf
```

This first assembly will use the lab estimates for the library sizes, but these are always off (sometimes significantly). The easiest way to update them is to edit the frg file to include the new values as computed by the assembly. The asm file will contain 1 estimate. The runca.amos.optereron script uses a different technique (created by Art) and the values will be written to the gbaf.lib.eval file. I update the frg file with 'vi', but you can also just update the gatekeeper stores. Since I updated the frg file (via symlink to pull directory), I deleted the entire directory and reran the entire assembly.

This assembly seems to be a bit better than my previous 0.2% error rate (the 4th largest scaffold is in 1 contig instead of 2), presumably because of the additional 60 reads.

UNITIG BLASTING

Unfortunately, both of these assemblies suffer from a relatively large number of surrogates and degenerate reads (~25% of all reads). The technique for reducing this is called unitig blasting. The idea is to take all surrogate and degenerate unitigs and blast them so that their reads are put into single read unitigs, and then rescaffold them. The hope is the scaffolder will be able to place the individual reads as stones, ideally to close gaps. An additional benefit is there will be no (or few) surrogate or degenerate reads. The tradeoff is after unitig blasting, all reads that are not in the scaffold will be singletons, and this can be an overwhelming number.

The procedure for this is as follows:

Create a report on all of the unitigs listing unitig id, scaffold status, and read

```
% mkdir /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/blasted
% cd /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/blasted
% /local/asmg/Linux_64/CA/bin/extractmessages IUM < ../gbaf.cgw | awk '{if
(/^acc:/){acc=substr($1,5);}else if (/^sta:/){sta=substr($1,5);}else
if(/^mid:/){mid=substr($1,5); print acc,sta,mid}}' > unitig.sta.read
```

Find all of the reads in degenerate or surrogate unitigs

```
% egrep 'N|S' unitig.sta.read > unplaced.unitig.sta.read
```

Find all of the unitigs that are degenerate or surrogate

```
% awk '{print $1}' unplaced.unitig.sta.read | sort -nu > unplaced.unitig
```

Remove the degenerate and surrogate unitigs from the cgb file

```
% strip-unis.awk unplaced.unitig < ../gbaf.cgb > strip.cgb
```

Assign a new unitig id to each read surrogate or degenerate read. The first read from each unitig is assigned the old unitig id, the other reads are assigned the first available unitig id. This is because Art believe the unitig ids must be sequential without any gaps.

First find the last unitig id in use:

```
% grep 'acc:' ../gbaf.cgb | tail -1
acc:1384
```

Now assign ids (free is assigned 1 more than the last acc), sorting by readid

```
% awk 'BEGIN{free=1385} {if ($1 == last){print free, $3; free++} else {print $1,$3;
last=$1}}' unplaced.unitig.sta.read | sort -k2,2 > newunitig.read
```

Extract the length of each read (clear range only) using internal CA ids (sort by readid)

```
% /local/asmg/Linux_64/CA/bin/dumpFragStore ../gbaf.frgStore/ | perl -ne '{if
(/readIdx:(\d+)/){$readIdx=$1}elsif(/Orig\((\d+),(\d+)\)/){$a=$1;$b=$2;$b-=$a;print
"$readIdx $b\n";}}' | sort -k1,1 > read.len
```

Join the new unitig ids with the read lengths
```
% join -1 2 newunitig.read read.len > readid.newunitig.len
```

(Make sure none of the reads have been lost, join is very sensitive to sort order, which depends a lot on the locale:
```
% wc -l readid.newunitig.len newunitig.read
```
should be the same)

Make the new unitig messages
```
% awk
'{printf("{IUM\nacc:%s\nsrc:\n.\ncov:0.0\nsta:X\nabp:0\nbbp:0\nlen:%s\ncns:\n.\nqlt
:\n.\nfor:0\nnfr:1\n{IMP\ntyp:R\nmid:%s\ncon:0\nsrc:\n.\npos:0,%s\ndln:0\ndel:\n}\n
}\n",$2,$3,$1,$3)}' readid.newunitig.len > new.cgb
```

Concatenate with the stripped cgb messages
```
% cat strip.cgb new.cgb  > gbaf.cgb
```

Link in the other assembly files
```
% ln -s ../gbaf.frg
% ln -s ../gbaf.frgStore
% ln -s ../gbaf.gkpStore
% ln -s ../gbaf.ofg
% ln -s ../gbaf.ovlStore
```

Resume the assembly with unitig consensus
```
% runAmos -C ~/bin/notes/runca.amos.opteron gbaf -s 90
```

This helped a bit- 2 gaps were closed and 300 more reads are in contigs. (See details below). It also helps to run cavalidate (and contig2fasta) at this time:

```
% mkdir /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/blasted/AMOS
% cd /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/blasted/AMOS
% ln -s ../gbaf.{frg,asm}
% cavalidate gbaf
% contig2fasta gbaf.contig > gbaf.fasta
```

MAIN CHROMOSOME IDENTIFICATION

The first step was to pull the 3 chromosome contigs from the database. The main tool for this is to use pull_contig and then use contig2fasta to convert to fasta format:

```
% mkdir /local/asmg/scratch/mschatz/GBAF/Chromosome
% cd /local/asmg/scratch/mschatz/GBAF/Chromosome
% pull_contig -D gbaf -A chromo_contigs -o chromo
% contig2fasta chromo.contig > chromo.fasta
```

Running ca2mates on the frg file will enable the matepair information to be displayed in the viewer on the 3 chromosome contigs:

```
% cd /local/asmg/scratch/mschatz/GBAF/2005-6-21/pull
% ca2mates gbaf
```

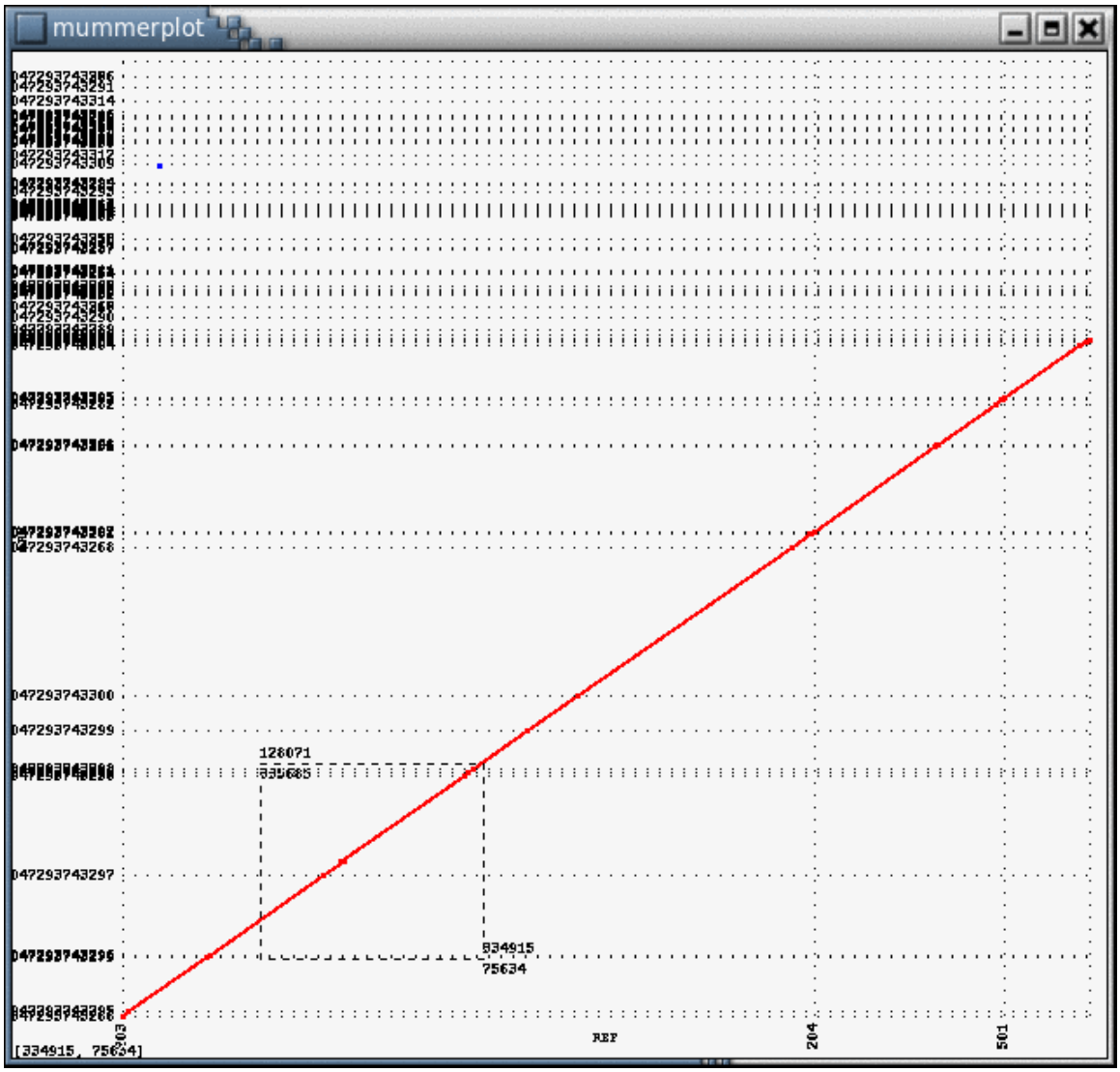
Now create an AMOS Bank of the 3 chromosome contigs:

```
% cd /local/asmg/scratch/mschatz/GBAF/Chromosome
% toAmos -c chromo.contig -s chromo.seq -q chromo.qual -m ../pull/gbaf.mates -o
chromo.afg
```

The first step now is to compare the new contigs to the old chromosome contigs. This is done by alignment with nucmer:

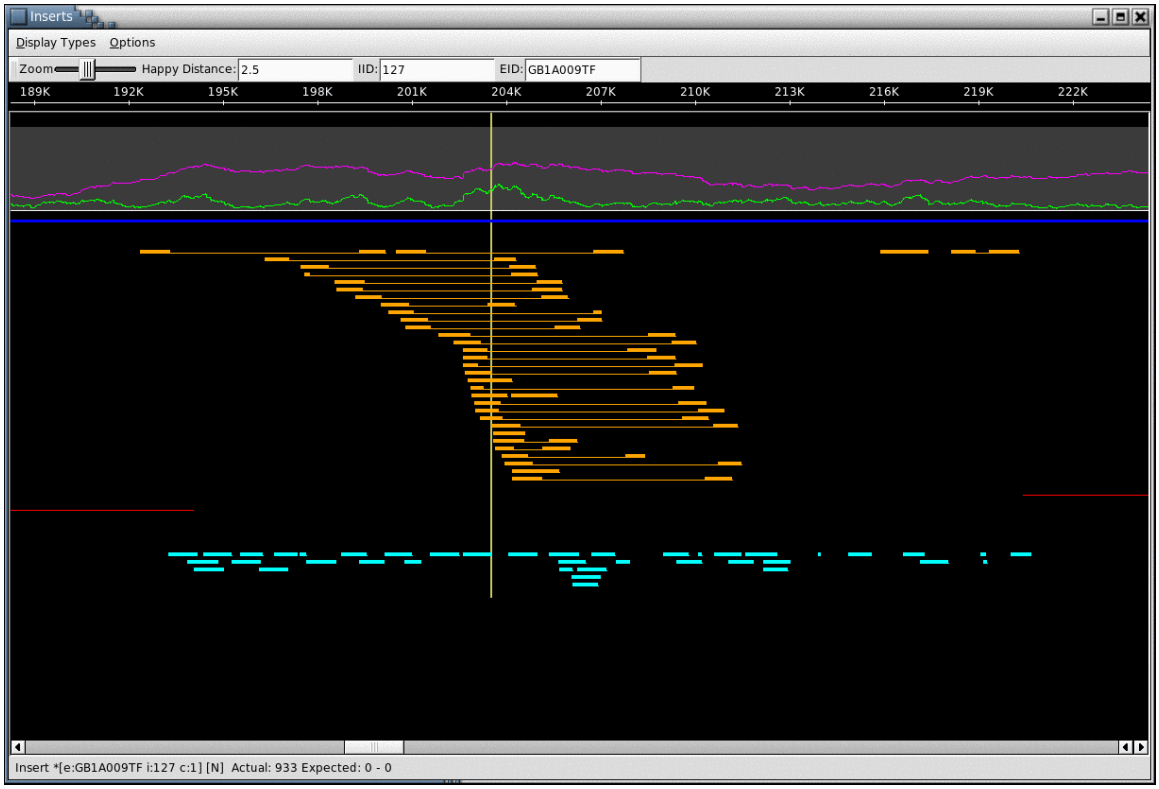
```
% cd /local/asmg/scratch/mschatz/GBAF/2005-6-21/0002/blasted/AMOS
% nucmer $LAS/mschatz/GBAF/Chromosome/chromo.fasta gbaf.fasta

Results:
% mummerplot out.delta -large –layout
```

Overall, the new assembly and 3 current chromosome contigs are in agreement. There is 1 major disagreement though:



This is strong evidence for a collapsed repeat in the reference (chromosome) contigs of ~3400 bp.

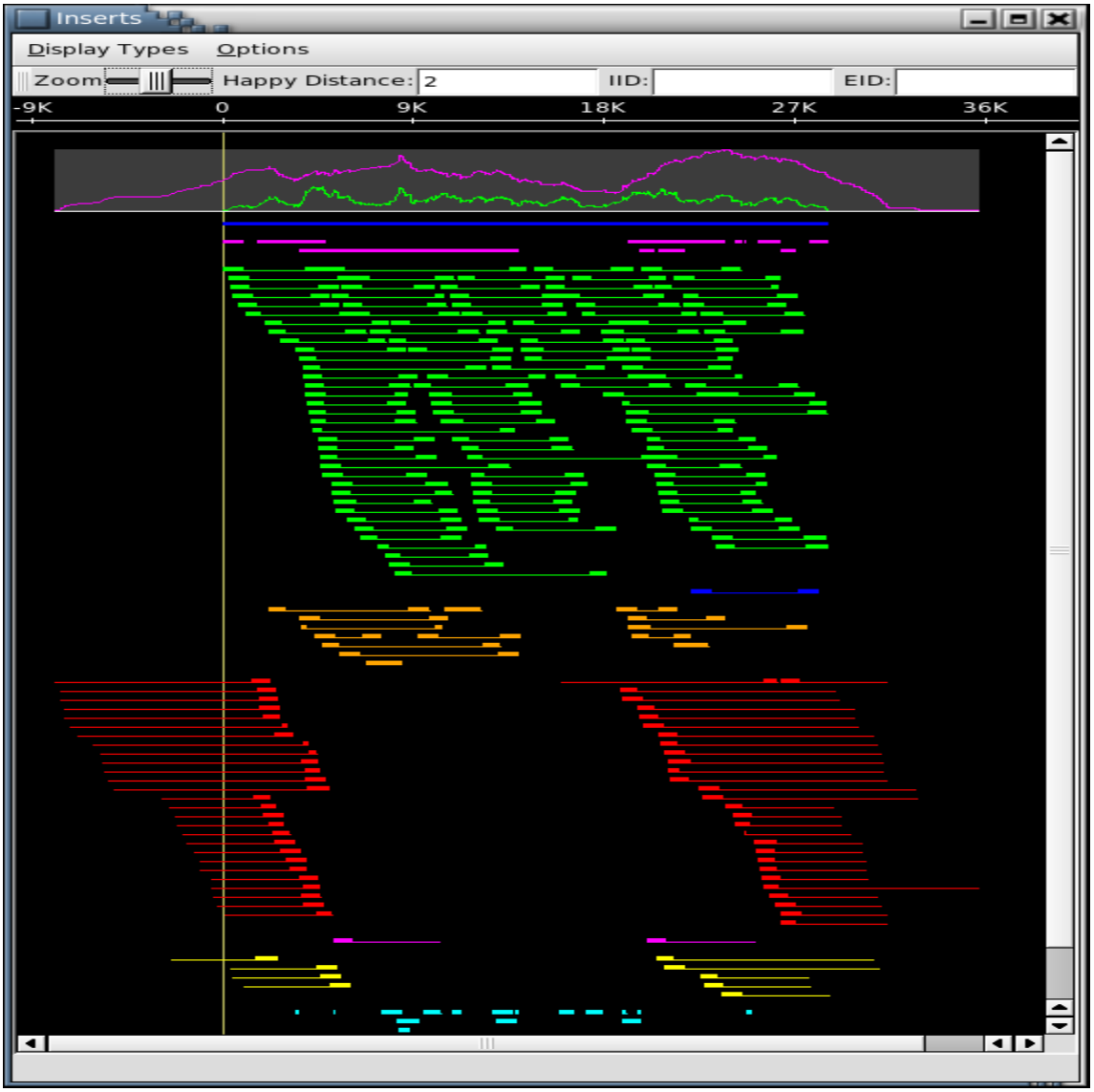
The large number of shrunken (orange) mates in this region of the reference assembly confirms this hypothesis.



PLASMID ASSEMBLY

The above alignment was primarily used to find disagreements with the reference chromosomes, but was also used to identify the reads which belong to the main chromosome. Presumably, if a read is in a contig which aligns to the chromosome, then the read is from the main chromosome, otherwise the read came from one of the unknown number of plasmids. In this way an initial set of plasmids reads were identified.

The identified reads were repulled raw separately (pullfrag -raw -N), and assembled using the same strict .2% error rate. This assembly was converted into AMOS banks as before (with cavalidate), and then the scaffolds were inspected by hand. A number of scaffolds were found to have a circular structure, meaning there were “misorientated” mates from the extreme left edge to the extreme right edge as show here (red mates are misorientated):



The reads from these circular scaffolds were assembled separately into the circles directory with the strict error rate. After the reads from a particular circle had been assembled separately, an assembly of the mixed plasmid reads was repeated, and the new scaffolds were inspected. This process was repeated until no circles were identified. These circle were aligned to the reference assemlies, and it was found that 1 circle almost exactly matches the plasmid that is shared with the two reference closed molecules. I also attempted to perform an AMOScmp assembly using the two reference sequences, but in the end they proved too divergent and CA performed better.

APPENDIX

QC Comparison of blasted vs non-blasted assemblies

	blasted	orig
[Scaffolds]		
TotalScaffolds	40	41
TotalContigsInScaffolds	72	75
MeanContigsPerScaffold	1.80	1.83
MinContigsPerScaffold	1	1
MaxContigsPerScaffold	12	14
TotalBasesInScaffolds	1283652	1346378
MeanBasesInScaffolds	32091.30	32838.49
MinBasesInScaffolds	1010	1010
MaxBasesInScaffolds	644814	641822
N50ScaffoldBases	644814	177956
TotalSpanOfScaffolds	1310241	1376341
MeanSpanOfScaffolds	32756.03	33569.29
MinScaffoldSpan	1010	1010
MaxScaffoldSpan	647207	647552
IntraScaffoldGaps	32	34
2KbScaffolds	28	28
2KbScaffoldSpan	1294424	1359335
MeanSequenceGapSize	830.91	881.26
[Top5Scaffolds=contigs,size,span]		
0	12,644814,647207	14,641822,647552
1	6,177956,178391	6,177956,178391
2	3,77102,78258	3,77102,78263
3	1,33134,33134	1,33134,33134
4	1,30134,30134	1,30134,30134
total	23,963140,967124,41875.65,221.34	
25,960148,967474,38405.92,366.30		
[Contigs]		
TotalContigsInScaffolds	72	75
TotalBasesInScaffolds	1283652	1346378

MeanContigSize	17828.50	17951.71
MinContigSize	1010	1010
MaxContigSize	199393	199393
N50ContigBases	68950	53254
[BigContigs_greater_10000]		
TotalBigContigs	27	28
BigContigLength	1162628	1217287
MeanBigContigSize	43060.30	43474.54
MinBigContig	10002	13100
MaxBigContig	199393	199393
BigContigsPercentBases	90.57	90.41
[SmallContigs]		
TotalSmallContigs	45	47
SmallContigLength	121024	129091
MeanSmallContigSize	2689.42	2746.62
MinSmallContig	1010	1010
MaxSmallContig	9502	9849
SmallContigsPercentBases	9.43	9.59
[DegenContigs]		
TotalDegenContigs	0	129
DegenContigLength	0	162566
MeanDegenContigSize	0.00	1260.20
MinDegenContig	0	120
MaxDegenContig	0	16082
DegenPercentBases	0.00	12.07
[Top5Contigs=reads,bases]		
0	1984,199393	1983,199393
1	1293,133285	1108,112488
2	1127,112488	1075,112064
3	1081,105518	1079,105518
4	685,72162	684,72162
total	6170,622846	5929,601625
[Surrogates]		
MinSurrogateSize	1189	914
MaxSurrogateSize	1189	18668
MeanSurrogateSize	1189.00	2766.79
SDSurrogateSize	0.00	3149.73
[Mates]		
ReadsWithNoMate	778	778
ReadsWithBadMate	308	342
ReadsWithGoodMate	7682	8178
ReadsWithUnusedMate	7914	7384
TotalScaffoldLinks	1	1
MeanScaffoldLinkWeight	6.00	6.00
[Reads]		
TotalReads	16682	16682
ReadsInContigs	11564	11264
BigContigReads	11025	10689
SmallContigReads	539	575
DegenContigReads	0	2012
ReadsInSurrogates	2	2463
SingletonReads	5116	943

[Coverage]		
ContigsOnly	7.32	8.32
ContigsAndDegens	7.32	9.52
AllReads	10.43	9.95
[gcContent]		
Content	0.28	0.28

Xanthomonas oryzae (Xoc)

April 2005

Seth performed the initial assembly on March 23rd using run_CA with default parameters. Art reran an assembly of this data with a 1.5% error rate for the unitigger using his tip of CA. This created two large scaffolds. He investigated this and found there were mates off the ends of the scaffolds into a degenerate contig:

Left Scaffold <=> Degen <=> Right Scaffold

I believe he found this by investigating the Contig link messages in the CA asm file, but the exact technique is unknown to me. He then manually edited the A-stat on the degenerate contig separating two large scaffold so that it would be placed in the scaffold, and reran cgw. This created a single large scaffold (as expected) with roughly 70 gaps.

Around this same time, I performed an assembly with Arachne. It performed better than Seth's but not as well as Art's. We sent this to Broad, and they sent back a better assembly, but not as good as Art's.

I also began to investigate and correct a number of likely collapsed repeats. These regions were identified by running '/local/asmg/Linux/bin/cavalidate prefix' where prefix is the prefix to the frg and asm files. This script is a wrapper for Mihai's asmQC mate happiness validator, and my own scripts for finding correlated SNPs. The assembly is converted into AMOS format for asmQC, and the features are written into the bank as well. The features are then clustered into a "prefix.suspicious.regions" file with contig range and aspects that are suspicious.

I found that I could correct the collapsed regions by performing a local assembly of the reads and mates in the regions using run_CA with default parameters. The strategy going in was to perform the local assembly, and then stitch the local assembly back into the original contigs, if it was successful. At some point, it occured to me that it would be significantly easier to replace the entire contig with a correct version, leading me to experiment with adjusting the error rate setting on the unitigger. I found that by performing a local assembly with the reads and mates in the contig in question with a strict error rate, a corrected contig was created, so there would be no need to stitch it back together.

The error rate determines how agreesively reads should be joined together into a single unitig- as a rule of thumb bigger unitigs are better, but a larger unitig has a higher potential for being an overcollapsed repeat. The trick is to find an error rate setting which separates the repeat copies into separate unitigs while not going so low as to break the contigs prematurely by ordinary sequencing errors. I eventually found that a .3% error rate (unitigger -e 0.003) was the balance for

this genome. The error rate is stored as an discrete value in .1% increments 0.0 -> 9.9. The standard run_CA using a 6% error rate; prior to this genome, no one had never gone lower than 1.5%. Art (always?) usually uses 1.5%.

The sweet spot error rate for a particular genome will vary by how repetitive the genome is, how much difference there is between repeat copies, the depth of coverage of the assembly, how well the trimming was performed, and how "clean" the clear range is. There is an error correction module in CA before unitigger that adjusts the error rate for a given overlap that is sensitive to the depth of coverage, so it is thought that this genome at 10x benefitted from this error correction. Art conjectured that its GC content was also beneficial, but it is unknown if what effect (if any) this really had.

From this work, I created a small (5) number of contigs that I was going to replace in the assembly which corrected collapsed repeats, but otherwised matched the original. I then entered "Data management hell" trying to replace the contigs: replacing the contigs was trivial, but I had to be sure to also fix the surrogates, features, degenerates, and scaffold with the new contigs. I got partially through this and decided as an experiment to run the assembly globally with the strict error rate. My expectation was that a small number of scaffolds (5-10) would form and the contigs would be more fragmented, and then we would have to decide if it was easier to fix the new scaffold or push through replacing contigs.

To my surprize, a single scaffold was formed with fewer gaps than ever- even fewer than by using the AutoJoiner on the prior best. I aligned this to the prior assembly and found that the new assembly resolved a number of over-collapsed repeats including beyond those that I had fixed though local assembly. The mates and correlated SNPs does indicate a few spots, but significantly better than any prior Xoc assembly.

Attempting to improve the assembly even further Art and I inspected the scaffold, including the reads and unitigs that could be placed inside the gaps (CA file prefix.gapreads). We found some of the larger gaps did in fact have reads that could be placed inside, but those reads were "trapped" in degenerate contigs. We then "blasted" all of the reads in degenerate or surrogate unitigs into singleton unitigs by creating the appropriate messages for cgw. We than rescaffolded using the original placed unitigs and the singleton unitigs. This was effective at reducing the mean size of the gaps from 148 to 81bp and the bases in scaffold increased by 5kb, but it did split 1 contig into two pieces (no gaps were closed in this process).

I then ran the CA backend with AutoJoiner on both the strict assembly and the blasted assembly and found that it did best on the blasted assembly. I ran AutoJoiner with a somewhat stricter criterion for joining contigs together because of the high GC nature, but it still closed 15% of the gaps with a mean gap size of 20bp. Friday morning Jason uploaded the assembly.

In my mind the default error rate in run_CA should be lowered, and it make senses to try multiple values for a particular genome. The cavalidate script could be run on every assembly, although I'm not sure how much benefit it would give, except as a report for Closure to regions to watch out for. As it currently is, Mihai's mate happiness marks every region where there are 2 or more compressed/expanded/misoriented mates but this should probably be adjusted to filter the noise (20 good and 2 bad is still marked). My scripts for finding correlated SNPs could be made smarter to take into account quality value to filter the false hits as well. The mate and snp features are written into the AMOS banks and can be viewed across the scaffold in my viewer, but it would be easy to convert them into a featUpload'able format for a contig-by-contig view in Hean's assembly viewer.

The experiment with blasting apart the the degenerate and surrogate unitigs is interesting, but had little effect as it was done. I now have some scripts for performing local assemblies based on the regions cavalidate detects, but we need better tools for updating assemblies before that can really be used effectively. Running Arachne on this dataset was useful experience, but in the end the CA assembly was better- even after Manfred spent some time on it (granted not a lot).

Trichomonas vaginalis (TVG)

January 2005

CA with overlap based trimming
AutoJoiner
AutoEditor
“Assemble in the gap” with minimus

See Powerpoint slides

Drosophilia Virilis

January 2005

This assembly was primarily carried out by a team at TIGR using the Celera Assembler, with significant participation by groups from the Venter Institute (VI) and the University of Maryland (UMD). This file contains a high-level recipe describing what we did.

1. Trim all reads for quality using Lucy, throwing away "shorts" (less than 64bp).

2. Trim all reads for vector using NUCmer (part of the MUMmer package), throwing away vector-only reads.

3. Retrim all reads to remove linker sequences, using Mihai Pop's 8mer counting program. Remove 8mers that are highly over-represented on the 5' end of sequences.

4. Trim the output of (3) further using the UMD overlapper and retrimming routines. For this step, these routines were used only to trim, not to extend reads.

5. Assemble using the Celera Assembler with "bubble smoothing" turned on. This is not a default option because it sometimes crashes.

6. Picking up the assembly from step 5 at a checkpoint following standard contig and scaffold construction using the cgw module, we made two additional passes over the assembly. The first (using the extendClearRanges module) attempted to close intra-scaffold gaps by extending
fragment clear ranges and allowing lower-quality alignments; 1159 gaps were closed in this fashion. The second (using resolveSurrogates) attempted to uniquely place individual reads from surrogate unitigs, based on mate pairs; 27,735 fragments were placed. (Surrogate unitigs are repetitive contigs, assumed to appear in more than one place in the final assembly. Although they are used to construct the final consensus sequence, the reads contained within them are initially not mapped to the consensus because they would appear in more than one place.)

7. Run Mike Schatz's AutoJoiner to close gaps. This closed about 5.5% of the intra-scaffold gaps that still remained after the steps above.

8. Recruit additional degenerate contigs to the assembly. Use NUCmer to compare
the 43,409 degenerate scaffolds (not normally included in the final assembly) to the "real" contigs. From the set of degenerates that didn't match the

assembly up to this point, we identified 23 that were greater than 2000bp in length and added them to the assembly. The largest was 13kb.

Some overall statistics:

[Scaffolds]
TotalScaffolds 1186
TotalContigsInScaffolds 7939

TotalBasesInScaffolds 165 Mbp (approx)
Max Scaffold Bases: 19,890,461
Max Scaffold Span: 20387473
N50ScaffoldBases: 8,064,686

TotalSpanOfScaffolds 179,596,666

[Top5Scaffolds=contigs,size,span]
388,19890461,20387473
346,16659257,17114172
189,13489559,13698929
500,11400460,11728403
158,9613644,9738782

[Contigs]
MaxContigSize 472205
N50ContigBases 69847

[BigContigs_greater_10000]
TotalBigContigs 2957
BigContigLength 149044481

[Top5Contigs=reads,bases]
7685,472205
6738,437639
6428,436539
6530,394065
5896,379600

Brugia (BRG)

January 2005

Subject: Brugia Stuff
From: "Delcher, Arthur" <adelcher@tigr.org>
Date: Wed, 5 Jan 2005 11:48:36 -0400
To: "Ghedin, Elodie" <eghedin@tigr.org>
CC: "Schatz, Michael" <mschatz@tigr.org>, "Koo, Hean L." <HKoo@tigr.org>,
"Schobel, Seth" <sschobel@tigr.org>

Hi Elodie,

I promised I would try to estimate the Brugia genome length and give
you a brief description of what I did to get rid of the most dubious
surrogates in the last assembly.

Both are below. Let me know if you need more explanation.

--Art.

ESTIMATING GENOME LENGTH

There are two factors that make it difficult to
estimate the length of the Brugia genome.

1. There are too many singleton reads.

Of the 176,099 singletons, 94,147 have no overlaps.
Of those 50,924 (54.1%) have GC-content >= 40%.
Of the 1,056,906 non-singleton reads, 50,088 (4.7%)
have GC-content >= 40%.
So the singletons without overlaps have a distinctly
different GC composition than the rest of the genome.

A plot of the GC content of reads in contigs, degenerates,
singletons, and singletons without overlaps is in
reads.gc.png (attached). The bump near 20% GC for
singletons and degenerates also looks suspicious.

I tried BLASTing a few of the high-GC singletons and found
31 matches to E.coli, and lots of matches to Ascaris (a
nematode) rRNA which may be similar to the Brugia rRNA, but
I don't think this accounts for anywhere near all the
singletons. For several of the reads the best BLAST matches
were to mouse sequence.

There are also a significant number of small (<=3 reads)
degenerate contigs with GC-content >= 40%.

I think there is either some kind of contaminant in the data,
or else the genome has distinct, high-GC regions that are
severely underrepresented in our shotgun sequencing.

Perhaps someone doing annotation could look into what
some of these sequences are.

2. Reads are not uniformly distributed in contigs.

If reads were sampled uniformly at random from the genome,
one would expect the coverage within contigs to peak
around the average coverage value. I.e., if the reads
cover the genome at 9x, then the depth of coverage in the
contigs should peak around 9x. The Brugia coverage
varies widely with large regions at very deep (>= 20x)
coverage, and the peak coverage of large contigs occurs
between 4x and 5x--well below the average coverage value.
File brg-tvg.coverage.png (attached) is a plot of the
coverage of large (>=5kb) contigs for Brugia, and also
for Trichomonas as a comparison. The average coverage of
Trichomonas is between 6x and 7x.

Another way to see this is in kmer frequencies. I counted
the number of occurrences of 22-mers in all the reads, and
plotted the percentage of distinct 22-mers that occurred once,
twice, 3 times, etc. One expects a spike for unique kmers
caused by sequencing error (any kmer containing a sequencing
error is likely to be unique), and then a peak near the
average coverage depth. File brg-tvg.kmerfreq.png (attached)
is a plot of the kmer frequency distribution for BRG and TVG.
TVG has the peak as expected; BRG has no peak.

Despite these problems, I tried to estimate the genome length
anyway.

1. From the assembly itself.

There are 70.7Mb in scaffolds and the total span is 77.5Mb. So
there are 6.8Mb in gaps. There are 17.5Mb in degenerates.
Using the degenerates to fill the gaps and then adding what's
left over would give a genome size of 88.2Mb. Adding the
singletons would make it way bigger--the total length of
singletons is 108Mb. I think the best estimate
from the assembly is between 80 and 90Mb.

2. From coverage of unique regions in the assembly.

Using frequent kmers to mask out repetitive regions of the
assembly we can compute the coverage of the remaining regions
and extrapolate this to estimate the genome length. Using
22-mers occurring >=40 times to indicate repeats, the
average coverage in remaining regions of contigs and degenerates
is 8.1x. If we divide the total length of all reads by
this coverage the implied genome length is 109Mb. If
we discount the singleton reads, the implied genome length
drops to 95Mb.

3. From the distribution of start positions of read overlaps.

Again masking out repetitive regions with frequent kmers

(this time 22-mers occurring >=30 times), I use a Poisson distribution to model the number of overlaps that occur in a given position of a read. The fit to the model isn't very good, but the resulting genome length is 88Mb.

Overall best guess: somewhere around 90Mb, but could be much larger if all those singletons really belong.

HOW FINAL ASSEMBLY WAS DONE

In the initial run of the Celera Assembler (CA), we observed a substantial number of large unitigs with coverage depth that was approximately double that of the majority of large unitigs. CA treats high-coverage unitigs as potential repeats and allows "surrogate" copies of them to be placed in multiple places in the assembly. In our assembly the large, double-depth unitigs typically occurred in exactly two places, on the ends of contigs, with mate pairs indicating that the two occurrences should instead be a single occurrence with respect to the neighbouring contigs.

To force these large unitigs to have a single occurrence in the assembly we identified all surrogate unitigs containing at least 50 reads, with at most two occurrences in the assembly, and all these occurrences on the ends of contigs. The coverage value of these unitigs was artificially reset to a value to force them to have a unique occurrence and the contigging/scaffolding stages of CA rerun.

Technical Notes:

Let S be the set of surrogate unitigs with >= 50 reads and <= 2 occurrences in the original assembly. Working in from both ends of each contig, choose unitigs that are in S, stopping at the first unitig not in S. Reset the a-stat coverage statistic of the unitigs to 6.01 and rerun the assembler starting at cgw.

The sequences (and IIDs on the header lines) of the boosted unitigs are in file boosted.utg.fasta . Their UIDs in the .asm file are in file r50.boost.uid and the connection between IIDs and UIDS is in file Promote-Surro/boosted.uid.iid .

Gi PolyDNA Virus (GPV)

February 2004

This project was a mixed sample of a highly variable genome that had been assembled together. The goal was to identify the variants by analyzing the snp patterns. I wrote a tool that uses graph coloring to try to identify sets of consistent reads, but it is very much incomplete.

The first step was to circularize contig 397, which was known to be circular because of mate pair relationships and alignment between the beginning and end of the contig. I pulled the contig from the database, and performed an alignment between the left and right edges of the contig. I found that the first 1622 bases aligned with the last 1590 bases at 96.73% identity. The disagreements consisted of mostly of SNPs scattered throughout the alignment, a 29bp insert in the left side that was not present on the right side, and a 3bp insert present of the left side but not on the right.

Using this alignment information I was able to circularize the contig by "zipping" the left side of the contig onto the right side. The "zipping" operation is performed by a still somewhat experiment tool I wrote called zipSlice, that can merge existing contigs together by using the alignment information generated by nucmer. zipSlice is the core of what will become the microassembler suite which will be used for assembling reads in the context of directed closure in the future. The process is "zipping" rather than "assembling", because the existing contigs are held "in-place" and alignment gaps are promoted to all of the underlying reads as a single operation. The circular contig after zipping has sequences that exist on both side of the assembly, because they cross the current origin (starting point of the linear representation). The contig file of the circular contig assigns negative offsets to the reads that cross the origin to indicate how far they "wrap-around" into the right side of the contig.

With the newly circularized contig, I zipped contig 235 to increase coverage. All of contig 235 aligned within the circular contig for ~1600bp at 98.38% identity. There was a small number of SNPs introduced between the circular contig and 235, and a 19bp insert in the circular contig that was not present in 235. Fortunately, the alignment with 235 occurred in the middle of the circular contig, so there were no issues with spanning the origin.

The next step was to reverse the contig so that it would have the same orientation as the reference. I performed this operation using my tool revSlice, which is a tool currently in production. Once the contig was

reversed, I rotated the contig so that its origin was the same as the reference. I determined the amount of rotation by aligning to the reference. I found the first 3928bp of the reference aligned to coordinates 14662-18591 of the contig and 3929-18528 of the reference aligned to the first 14662bp of the contig, indicating I needed to rotate the contig so that position 14662 was the new origin. I performed this with another tool I wrote called rotateSlice which in essence shifts the tiling from the left side of a contig to the right side of a contig a specified amount. Because the contig is circular, this is effectively a counter-clockwise rotation.

The final step is to recall the consensus. The contig that I had been working with had a number of ambiguity codes in it because it had been called using the "Conic Ambiguity Model", which compares the ratio of quality values between the elements in the slice to determine if the consensus is ambiguous or not. For purposes of alignment to the reference, it is more useful to not display ambiguity codes, since the reference has none and they would show up as false mismatches. I therefore recalled the consensus using a non-ambiguous model, which picks the consensus as whichever element (base or gap) has the most quality value associated with it in the slice. This was the original consensus caller for cloe several months ago, but was replaced as calling the consensus became a better understood problem. This was performed by another slice tool called, trSlice, which is useful for translating the coordinates of slices, but can also be used to just recall the consensus in place. I then converted that slice file to contig format using slice2contig (another Slice Tool), and converted that contig file to fasta format with contig2fasta, giving the final result of recall.fasta.

The SNP report that I generated works off of the tcov format of the contig which shows the tiling at each location. The tcov format was generated using getCoverage in tcov mode (--tiling). The report identifies each consensus position where there is at least one disagreement between the reads that tile that location. For each position that it found, it reports the count of the base calls for and against the consensus, and now the read ids for each base in the slice. Of the 18595bp of the consensus, my tool identified 441 sites with at least one disagreement. Preliminary analysis of this report does show a strong correlation between the disagreements for the large insert regions. The tool that generated the report is written in perl, and can easily be extended as we gain better understanding of identifying correlated disagreements and genome variants.

Cyanobacteria (GYMA & GYMB)

September 2004

Luke found that the assembly of GYMB looked bad and there were several mistakes made by AutoEditor. I was asked to see if I could figure out what happened. In the end it was determined that there was a tracking problem and two different strains of the organism had been assembled together.

The first attempt was to try running the consensus caller from the updated pre-production Celera Assembler (CA3), to try to see if the bugs fixed in the consensus caller would fix the problems we are seeing. I ran it, and it did perform a somewhat better job and left fewer gaps in the assembly. For details compare /local/asmg_scratch/mschatz/GYMB-CA3Cons/bubbleReport, which shows at worst there were 2 reads that had 24 gaps in a row with the new consensus caller, versus /local/asmg_scratch/mschatz/GYMB-ORIG/bubbleReport which shows at worst there were 13 reads with 353 gaps in a row in the original assembly.

However, while the number of gaps drop dramatically, it really only masks the problem and replaces long stretches of gaps with really poor alignments. For example:

```
#GYXAB25TF(577376) [RC] 923 bases, 00000000 checksum. {807 33} <576164 577082>
...
CCTTGTCAACTGGATTGGGGTTTGTGCCTGGAAATGCTTTTTCTCTCAAGCGGAGCCCTG
AA-----------------GG-ATGCAGTT-----T--TCC-T--GCT-C-C----CCTC
T------T--CCC---TT-ATG-G---G-AA---A--G--G-----GG-T----T--G--
-CCT--G----T-----AAG-CA---G-CGCA-GAACC-TTGG---GCAA-G-TG---C-
----CGC-A--AA---------T-T----G-A--G--C----TT-AACT----T-A-C--
--TTGTCGATGCCTTCCT-GTCAATGCCCCA-CCGAGATCTCCTACAACT---AGTTTGG
```

The second attempt was to try to run the entire assembly with CA3. This produced the fewest gaps in a row overall (/local/asmg_scratch/mschatz/GYMB-CA3/bubbleReport), but has dramatically more contigs (1041 in scaffolds vs 664) of smaller size (6179 N50ContigBases vs 375173). It appears that CA3 recognized the contamination problem, and rather than incorrectly forming contigs with distorted regions, left the assembly "shattered". While this assembly is probably more accurate that the original assembly in some sense, it has nearly

50% more scaffolds and over twice as many sequencing gaps.

Our next thought was to try to identify the reads which are causing the problems, and reassemble without them. I created a report on all high quality conflicts in the entire original assembly (/local/asmg_scratch/mschatz/GYMB-ORIG/gymb.snps). From this, I created a report on the number of high quality conflicts each read has with the consensus, i.e. the number of times it has the minority basecall (gymb.minority). This shows that there are 3074 reads which have at least 1 high quality conflict with the consensus. Obviously, a good number of these are due to simple base calling errors, so we cannot use the entire list. However, there are reads that have as many as 365 conflicts with the consensus that should definitely be excluded from the assembly.

I've created 2 more reports, gymb.minority.10 & gymb.minority.5 which have the reads names with at least 10 and 5 conflicts with the consensus, respectively. The minority 10 report has 716 reads, minority 5 has 1216. A new report at any level of conflicts can easily be created (awk '{if ($2>=5) print $1}' gymb.minority > gymb.minority.5) If we can believe quality values at all, there should only be a very small number of high quality base calling errors per read,so presumably all other conflicts are because of cross contamination between projects.

My best recommendation is to look at the minority reports, decide on a threshold of the maximum number of conflicts with the consensus that you are willing to accept, exclude the reads beyond the threshold from a new frg file, and reassemble. Fortunately, there aren't a lot of reads in the project, so it will be relatively quick to assemble, possibly trying out a few thresholds simultaneously. If you want, I can inspect the new assemblies, and generate new reports on gaps and conflicts.

Once we get an assembly that we have more confidence in, we can begin to work on the AutoEditor problems. There are a few unexposed parameters that I can adjust which will make it use a stricter policy for making edits that will hopefully prevent it from making invalid edits on any remaining contaminating reads.

Luke Responded:

i'd like to have gymb reassembled while excluding the reads in gymb.minority.5 (1216 reads). i'll submit a ticket to data support for this.

i understand why autoeditor could make such errors when presented with such poor alignments and distorted traces. however, the root of the problem then seems to be the CA alignments. i would still like an explanation as to how CA

could produce such poor alignments and how will we detect and prevent/correct these situations in the future.

And then later:

Mike has reassembled gymb after excluding the 1216 reads from the minority.5 file. the results look pretty good. 99% is in four contigs in one scaffold. a nucmer between the old contigs and the new ones shows almost perfect recreation of the genome order/orientation, etc.

do you think you could take a look at the new assembly and generate the same reports on conflicts and gaps, etc? hopefully we'll see some much lower numbers.

My Final Response:

Luke,

I'd say the assembly is in much better shape with the contaiminating reads removed. I ran the snp and bubble reports on both the original assembly (in /local/asmg/scratch/mschatz/GYMB2/orig) and the autoFinish'ed assembly, ie with autoEditor + autoJoiner (in .../autoFinish).

There are still 1157 high quality discrepancies in the original assembly (gymb.snps), including reads that are in the minority a maximum of 32 times (gymb.minority). Fortunately, there are only 32 reads with 5 or more discrepancies. I looked at some of these, and found most had been in very small contigs or singletons in the previous assembly, and consequently weren't picked up as being contaminates before.

In terms of gaps in a row, the assembly is drastically better off. There was a maximum of 4 gaps in a row (bubbleReport) which is much more reasonable than the 10s or 100s we had been seeing. I'm also encouraged to see that in the post-autoEditor snp report there are still 1115 discrepancies. This means that autoEditor only edited on the order of 50 high quality bases, which is a much more reasonable number and suggests that it didn't get caught making edits where it should not have been. I believe this to be because of the drastically better alignment quality.

At this point you have a few options-
1) Perform another reassembly with the 32 latest reads filtered out. This is the safest move, but may not have much effect.

2) Upload the pre-autoFinished assembly. This would give you an opportunity to look at the assembly pre-autoEditor and maybe get a better sense of how widespread the contaimination still is. This also avoids any possiblity of

errors introduced in post-assembly operations.

3) Upload the post-autoEditor assembly. While the number of edits that autoEditor made was pretty small (< 10% of all discrepancies including low quality), it will hopefully save you team some work.

4) Upload the post-autoFinish assembly. This includes, in addition to the autoEditor results, the impact of the autoJoiner. I looked at it briefly, and autoJoiner was able to close 6 sequencing gaps, and joined some of the large contigs so that there is just 2 (see .../kmoffat/ASM/GYMB2/asm2004_0824/autoFinish/gymb.joinqc and .../gymb.joinreport). I fully acknowledge that the consensus may be a litte rough using bases outside of the clear range, but it might help the closure efforts if you can see a rough draft of the consensus and know within a few bases the exact size of the gap. The autoJoiner regions will be highlighted in the assemblyViewer, so you'll know exactly where to find them.

As for why did CA3 perform so different than CA2, I am still investigating it closer. It's not clear to me yet if CA2 is simply more agressive than CA3 or if there are simply bugs, but I can say that CA3 does tend to be a bit more conservative in the assemblies that I have studied. I'm currently performing a study of CA2 versus CA3, and hope to better understand this exact issue better in the near future. I'll let you and everyone know what I turn up.

I have full confidence that the data support team will be able to help you uploading/reassembling, but please let me know if there is anything else I can assist with.