

Winter 2009

Comparative genomics exploration tools

Shilpa M. Kulkarni

University of New Hampshire, Durham

Follow this and additional works at: <https://scholars.unh.edu/thesis>

Recommended Citation

Kulkarni, Shilpa M., "Comparative genomics exploration tools" (2009). *Master's Theses and Capstones*. 524.
<https://scholars.unh.edu/thesis/524>

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact nicole.hentz@unh.edu.

COMPARATIVE GENOMICS EXPLORATION TOOLS

BY

SHILPA M. KULKARNI

Bachelor of Engineering, Maharashtra Institute of Technology, 2000

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science
in
Computer Science

December, 2009

UMI Number: 1481728

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

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



UMI 1481728


Copyright 2010 by ProQuest LLC.

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

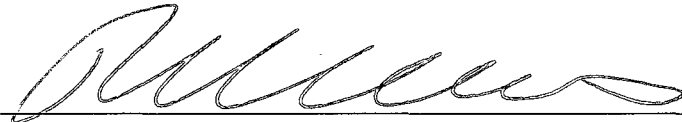


ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

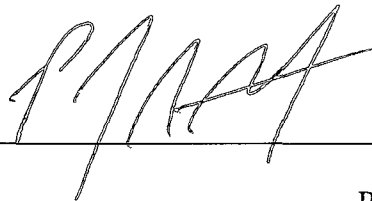
This thesis has been examined and approved.



Thesis Director, R. Daniel Bergeron
Professor of Computer Science



W. Kelley Thomas
Professor of Biochemistry and Molecular Biology and Genetics



Philip J. Hatcher
Professor of Computer Science

12/14/2009

Date

ACKNOWLEDGEMENTS

I take this opportunity of expressing my heart-felt gratitude towards the key personnel who have been instrumental towards the completion of this thesis work: Dan Bergeron, W. Kelley Thomas and Phil Hatcher. I would like to thank Dan and Phil for encouraging me to take the Genomics courses as part of my Masters' course work. This led to my research work in the same area. Dan has provided valuable insights in the project and has tirelessly guided me throughout the research. I would like to thank Kelley for his immense knowledge, his interest in this project and for funding my research. My understanding of Genomics field would not have been easier without the weekly discussions carried out during the Genomics course taught by Kelley.

I would like to thank Krystalynne Morris for her immense support and the discussions I had with her about my thesis work. Her thorough understanding of the research area helped me enormously. I would also like to thank all my colleagues at the Thomas Lab for all their help: Darren Bauer, Abraham Tucker, Way Sung and Mingju Li.

I would like to thank my parents, my family and my friends for their constant support and encouragement. Last but not the least, I thank my husband Abhijeet and daughter Trishna for supporting and understanding me throughout the completion of this thesis.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iii
TABLE OF CONTENTS.....	iv
LIST OF FIGURES	vi
LIST OF TABLES	vii
ABSTRACT.....	ix

CHAPTER	PAGE
INTRODUCTION.....	1
BACKGROUND.....	4
MUTATION DETECTOR (MD)	7
3.1 Introduction.....	7
3.2 Motivation.....	8
3.3 Approach.....	10
3.3.1 Phase I, step 1: Creation of “Delta” file.....	11
3.3.2 Phase I, step 2: Alignment Summary.....	13
3.3.3 Phase I, step 3: Processing Alignment Summary	14
3.3.4 Phase II, step 1: Combining MD output files	16
3.3.5 Phase II, step 2: Applying coverage restrictions.....	18
3.3.6 Phase II, step3: Creation of analysis tables.....	20
3.3.7 Errors/Matches.....	21
3.3.8 Reference Difference Positions.....	27

3.3.9	Mutations	27
3.3.10	Putations (Putative Mutations).....	28
3.4	Implementation	29
3.5	Results.....	29
3.5.1	Base Substitutions.....	33
3.5.2	Reference Difference Positions.....	34
3.5.3	Heterozygous Sites and Mutations.....	34
MULTIPLE ASSEMBLY VIEWER WITH ENHANCED NAVIGATION		
(MAVEN).....		36
4.1	Introduction.....	36
4.2	Motivation.....	37
4.3	Existing Tool – BankViewer	38
4.4	MAVEN as an Assembly Test and Tuning Tool.....	39
4.5	Results.....	40
CONCLUSION AND FUTURE WORK		46
5.1	Mutation Detector	46
5.2	MAVEN.....	47
5.3	Iterative Multi-Genome Comparative Cross Assembly Processor	47
LIST OF REFERENCES.....		49
APPENDIX A		51
MUTATION DETECTOR (MD) DOCUMENTATION		51
APPENDIX B.....		60
MAVEN DOCUMENTATION.....		60

LIST OF FIGURES

Figure 3.2-1 Role of mutation and selection on genomic evolution.....	8
Figure 3.2-2 Reduced efficiency of natural selection	9
Figure 3.2-3 Relation between the reference sequence and the yeast strains.	10
Figure 3.3-1 Sample Delta file.....	12
Figure 4.3-1 BankViewer showing one (blue) contig and many (green) reads.....	38
Figure 4.5-1 Reads discarded from contig formation step in the assembly.....	41
Figure 4.5-2 Parts of a read aligning at different positions in the reference sequence.....	42
Figure 4.5-3 Reads with gaps accepted in contig formation.....	43
Figure 4.5-4 A view of <i>de-novo</i> assembly results.....	44
Figure 4.5-5 A MAVEN display of results from a comparative assembly.....	45

LIST OF TABLES

Table 3.3-1 Sample Alignment Summary.	13
Table 3.3-2 Removing the insertion positions.	14
Table 3.3-3 Homopolymer region.....	15
Table 3.3-4 Eliminating 5 base positions surrounding homopolymer region of 4 or more bases.....	15
Table 3.3-5 Sample MD output file for strain A1.....	17
Table 3.3-6 Sample MD output file for strain A4.....	17
Table 3.3-7 Sample MD output file for strain C5.....	17
Table 3.3-8 Sample MD output file for strain C8.....	17
Table 3.3-9 Sample Combined Output file for pair-wise analysis of strains A1 and A4.	18
Table 3.3-10 Sample Combined output file for all-strain analysis A1-A4C5C8.....	18
Table 3.3-11 Applying coverage restrictions for pair-wise analysis	19
Table 3.3-12 Applying coverage restrictions for all-strain analysis	19
Table 3.3-13 Consensus creation	21
Table 3.3-14 Calculation of Errors and Matches for a base position.....	22
Table 3.3-15 Sample data for 10 reference positions	23
Table 3.3-16 Matrix position calculation.....	24
Table 3.3-17 Error Matrix.....	25
Table 3.3-18 Match Matrix.....	26
Table 3.3-19 Reference Difference positions	27

Table 3.3-20 Mutations.....	28
Table 3.3-21 Putations	28
Table 3.5-1 Assembled positions in Genomic Assembly Vs. Chromosome Assembly ..	30
Table 3.5-2 Pair-wise strain analysis Vs. All-strain analysis.....	31
Table 3.5-3 Analysis tables for Genomic Assembly vs. Chromosome Assembly	32
Table 3.5-4 Yeast MA line mutations.....	33

ABSTRACT

COMPARATIVE GENOMICS EXPLORATION TOOLS

by

Shilpa Kulkarni

University of New Hampshire, December, 2009

Comparative Genomics focuses on elucidating the genetic differences between different species or different strains of the same species by the comparative analysis of DNA sequences to identify functional elements and regulatory regions. This thesis describes the design and development of two software tools to support comparative genomics research. These tools were specifically developed to support the analysis and assembly of sequence data produced from innovative new DNA sequencing technology from 454 Life Sciences using the PicoTiterPlate™ device. This technology will dramatically affect comparative genomics research. Currently available software tools were developed to handle traditional shotgun sequences averaging 500-1000 base pairs in length. These tools are inadequate to handle the unique characteristics of sequence reads generated by 454 Life Sciences. The goal of this research is to adapt currently available tools and develop new tools to be used for sequence reads generated by any sequencing technology, even those having different characteristics from the traditional shotgun sequences.

CHAPTER 1

INTRODUCTION

Comparative Genomics looks for the similarities and differences between species or different individuals within a species. It also looks at heterozygosity within a single individual. This can help us understand the effects of natural selection acting on the individual genomes, which forms the basis for understanding evolutionary process and how species evolve. Comparative studies of model organisms such as yeast and *Daphnia* can aid in understanding the process of evolutionary change in a well-characterized system, which can then be extended to other organisms. Comparative genomics relies upon software tools which can look for similar regions between genomes by lining them up against each other. Currently, these tools are largely designed for the traditional shotgun sequence reads. The new sequencing technologies produce reads that have very different properties from the shotgun sequence reads.

We developed two software tools and studied the feasibility of one tool with the specific goal to advance comparative genomics research with data from this new sequencing approach:

1. Mutation Detector (MD) – This tool aids in confirming the mutations between comparatively assembled genomes of closely related organisms. It can identify mutations unique to a specific strain as compared with the reference genome. It

can be used to analyze the genome assemblies produced by the AMOScmp comparative assembler [10]. Although this tool was motivated by and is optimized for the sequence reads generated by the sequencing technology developed by 454 Life Sciences, it has proven to be an effective tool for analyzing shotgun sequence data as well.

2. MAVEN (Multiple Assembly Viewer with Enhanced Navigation) – This genome assembly visualization tool assists interactive visualization of sequence assembly and alignment results for multiple genomes simultaneously. It is able to display the results of *de novo* assembly, comparative assembly, and sequence alignments at the same time. It aids in understanding how multiple strains of an organism assemble to a common reference sequence simultaneously.
3. Iterative multi-genome comparative cross assembly processor – The idea behind this tool is that it helps in simultaneous comparison of multiple genomes that are assembled initially using a *de novo* assembler. The contigs of one organism can then be used as a reference sequence and the contigs of the other organism can be comparatively assembled against this reference. The hope is that simultaneous cross assembly of the sequence data from multiple closely related genomes will allow us to produce a better assembly than separate *do novo* assemblies. During our investigation we found that the *de novo* assembly of the individual strains was only able to generate contigs that averaged about twice the size of the input sequences. We were not able to significantly increase the contig sizes using cross assembly either. This has been a major setback and has hindered the development of this tool completely.

In the sections that follow, background for the research is provided followed by a description of the two completed tools: Mutation Detector and MAVEN.

CHAPTER 2

BACKGROUND

454 Life Sciences has developed a revolutionary technology, producing fast and cost-effective DNA sequences using efficient, simple and convenient means. It is able to generate millions of raw bases within an hour on a single instrument. Based upon the genome size, a researcher can take less than a week to generate sequence reads and then to assemble them using their own Newbler Assembler. Using the shotgun process, it could take months to do the same task. This technology has very recently become available and promises to dramatically reduce the cost of genome sequencing.

The sequencing technology using the PicoTiterPlaceTM Device [7] from 454 Life Sciences can generate sequence data for 1% of the cost of the traditional techniques. This new technology promises to revolutionize the very important field of *comparative genomics*, which focuses on discovering the basics of evolutionary dynamics by comparing the genomes of different species. Because of the high cost of traditional sequencing, most comparative genomics studies so far have compared the genomes of species that are very divergent in evolutionary terms. As sequencing costs decrease dramatically, it is becoming feasible to compare the genomes of many very closely related organisms, which will yield more evidence to understand basic evolutionary principles.

The sequence reads produced by 454 Life Sciences have very different characteristics from the traditional shotgun sequencing reads. The 454 sequence reads are much smaller (averaging 108 base pairs in length) compared to the shotgun sequences that average 500-1000 base pairs in length. The traditional sequence reads have highly variable quality and the quality deteriorates at the ends of the sequences. Traditional sequences also have mate-pairs; the shotgun pieces of DNA are sequenced from both ends to produce the mate-pairs. As opposed to this, the 454 sequences have very high quality base calls, except for homopolymers and did not have mate-pair sequences when this research was carried out. This technology has a significant limitation in that it cannot identify the accurate length of homopolymers (more than 4 consecutive identical base pairs).

Effect on Sequence Analysis and Assembly Software

Existing computer-based sequence analysis tools were developed to handle the traditional shotgun data and have been designed and optimized to deal effectively with its characteristics and idiosyncrasies. These tools do not work particularly well with the data generated by these new sequencing technologies.

This newly emerging sequencing technology is bound to increase the number of organisms that can be sequenced within a short time. This, in turn, is going to place a high demand on the assembly processes, which reconstruct the original sequence from the smaller sequence reads. We need to produce more accurate sequence assemblies in less time. The results of these processes are crucial, since the annotation of the sequenced and assembled genome depends entirely on the assembly process. The mutation detection methods depend heavily on the results of the assembly process. Hence, choosing the appropriate assembler to assemble the sequences is imperative. The assembler needs to

be customized to handle the sequence reads produced by different sequencing technologies.

Comparative Assembly

In the comparative assembly process, the sequence reads are aligned by comparing them to a reference sequence. For all our comparative assemblies, we have used the tools provided by the AMOS package available for download at <http://amos.sourceforge.net/> . This assembler is called AMOScmp [10]. AMOScmp can also assemble sequences against multiple reference sequences simultaneously. For this, the multiple references need to be provided in a multi-fasta file format. AMOScmp uses the MUMmer alignment tool [1] to produce alignments. The aligned sequences that overlap each other are then used to construct the contigs and the consensus of the contigs produces the output for this process.

CHAPTER 3

MUTATION DETECTOR (MD)

3.1 Introduction

Mutation plays an important role in the evolution of organisms. All genomes are subject to a wide spectrum of naturally occurring mutagenic activities like recombination, replication errors, base substitutions and indels (insertions and deletions) [3, 6]. Although a small fraction of the mutations are useful for the survival of organisms under varied conditions, most mutations are considered to be harmful. Beneficial mutations are passed on to the surviving individuals in the next generations. Natural selection and DNA repair pathways reduce or filter mutations such that very few of these mutations are passed on to the next generation.

Mutations are very rare; the best current estimate is 1 in 10^9 mutations per site per generation. In reference alignment based assays, mutations must be separated from sequencing errors and alignment errors. The MD tool has been developed as a collection of programs and scripts that aid in confirming the mutations between the comparatively assembled genomes of organisms. This tool is used to analyze the genome assemblies produced by the AMOScmp comparative assembler [10]. It can be used for the sequence

reads generated by 454 Life Sciences using the PicoTiterPlate™ device [7] as well as for shotgun sequence reads.

3.2 Motivation

Until now, the study of evolutionary process by detection of mutations has been derived from indirect inferences:

1. phylogenetic comparisons of DNA sequences [5]
2. indirect observations of the behavior of artificial reporter constructs [2]

The first method makes assumptions about the times of divergence between the species and the neutrality of the surveyed genomic regions. The second method of inference makes assumptions about the mutation detecting ability in extrapolating to the genome level. Because of this, the indirect methods of mutation detection are inadequate. A new and more direct approach is to produce Mutation Accumulation (MA) lines that are developed in such a way as to effectively eliminate the ability of natural selection to eradicate mutations prior to their detection.

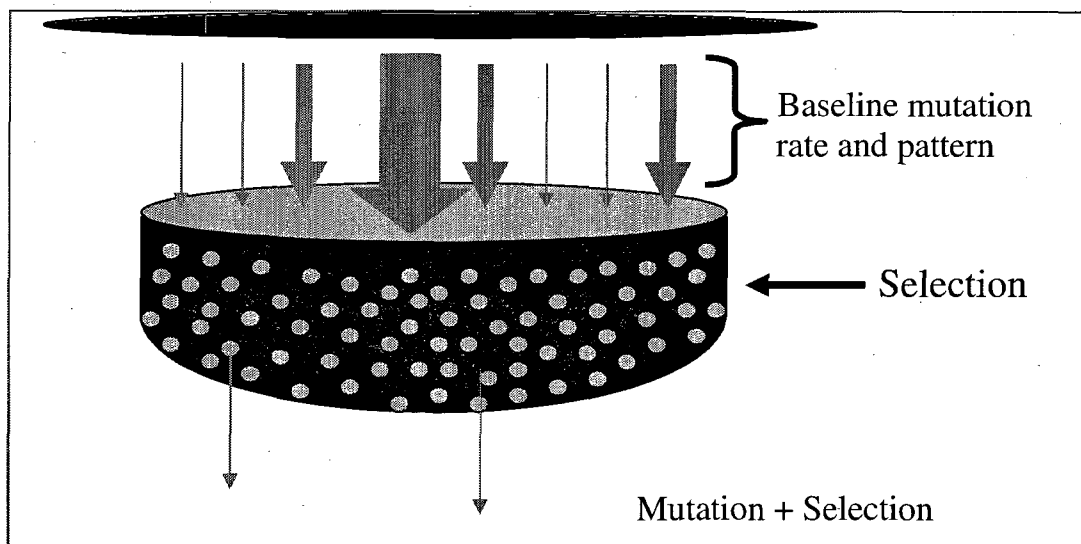


Figure 3.2-1 Role of mutation and selection on genomic evolution.

Figure 3.2-1 shows how natural selection eradicates the baseline mutations even before they are detected. The selection sieve does not let all the mutations pass to successive generations.

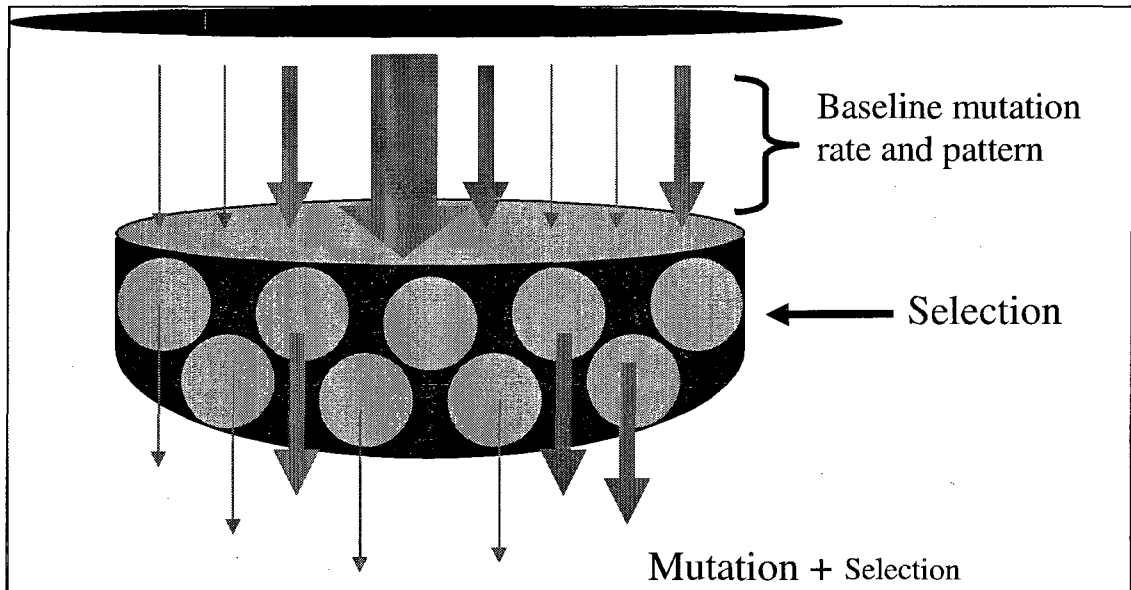


Figure 3.2-2 Reduced efficiency of natural selection

We worked with yeast MA lines that have been developed from a single colony of yeast *S. cerevisiae*. A colony is recognizable only after 5 generations have been grown from a single yeast cell. Initially, a single colony is randomly chosen to start the next generation. After five generations, another colony is randomly chosen and a new *mutation accumulation (MA)* line is started from it. This process is repeated for each MA line. Currently, the yeast MA lines have been propagated 4600 generations. Altogether, 48 yeast MA lines have been developed in the laboratory. From these, 32 are haploid and 16 are diploid. For this assay, we have been working with four haploid MA lines A1 (G1), A4 (G2), C5 and C8. Figure 3.2-2 shows how this random selection of colonies bypasses the normal evolutionary process of natural selection that tends to limit the accumulation of deleterious mutations and linked neutral mutations.

The approach is used to detect mutations across these MA lines and thus study the evolutionary process. For mutations to be effectively studied in evolutionary genomics, detection of hundreds of mutations is necessary. The conventional PCR amplification and direct sequencing methods traditionally used for this purpose are not as cost-effective as the highly parallel sequencing technology developed by 454 Life Sciences. The highly parallel micro-technology is the most cost-effective for genome-wide mutation discovery. The availability of these sequence reads has been a motivating factor in the direct analysis of mutation rates and identifying qualitative aspects of the mutation spectrum.

To study the mutation along each yeast line, it is compared with a reference sequence. The complete assembled sequence of *S. cerevisiae* is used as the reference sequence. In our case, we studied 4 MA lines A1 (also known as G1), A4 (also known as G2), C5 and C8.

Figure 3.2-3 shows the relationship between *S. cerevisiae* and the strains A1, A4, C5 and C8.

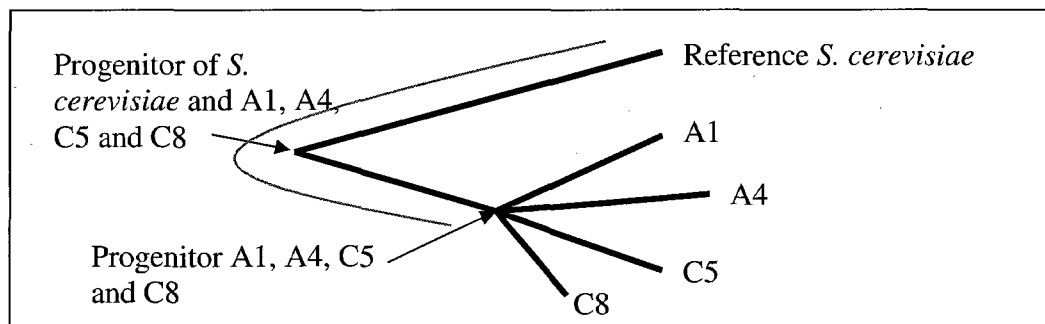


Figure 3.2-3 Relation between the reference sequence and the yeast strains.

3.3 Approach

The MD approach is carried out in two phases. Phase I is carried out for each of the individual MA lines. Phase I includes:

1. **Creation of the “Delta” file** - Comparative assembly of the MA line against the reference sequence *S. cerevisiae* produces an output file that shows read alignments against the reference sequence. This is called the “Delta” file.
2. **Alignment Summary** - MD scripts produce an alignment summary of the sequence reads against the reference sequence.
3. **Processing Alignment Summary** – The alignment summary that is produced in step 2 is processed to remove insertions and homopolymer regions (regions that have more than 4 consecutive identical bases – any one of Adenine, Thymine, Guanine or Cytosine).

Phase II of the approach is carried out for multiple MA lines and includes three steps:

1. Combining MD output files
2. Applying coverage restrictions
3. Creation of analysis tables

3.3.1 Phase I, step 1: Creation of “Delta” file

We used the AMOScmp assembler to assemble the sequence reads for the MA lines using as a reference the complete assembly of the *S. cerevisiae*. MUMmer [1] performs the alignment step of the assembly and produces a ‘delta’ file that gives the alignment information for each read with respect to the reference sequence. It also tells about the relative distance between insertions and deletions in the alignment. Figure 3.3-1 shows a small portion of alignment data stored in a sample delta file. MUMmer internally uses NUCmer (NUCleotide MUMmer) to produce the alignment data. The first line specifies that the data in the file corresponds to alignments between the reference sequence stored in the file ‘chr1.1con’ and the query sequences stored in the file ‘chr1C5.seq’. The

second line says NUCmer was used as the alignment script. The alignment information between each query sequence and the reference sequence starts with a header '>' followed by the reference sequence length and the query sequence length. The lines that follow denote the actual alignment information: start and end co-ordinates in the reference and the query sequences, the number of errors, the number of similarity errors and the stop codon (always '0' for DNA alignments). The header is followed by a string of signed digits, one per line, denoting the distance to the next insertion (positive integer) or deletion (negative integer) in the reference sequence. The final line for each alignment before the next header is '0'.

```
/home/bioinfo1/Yeast1/chr1/chr1.1con /home/bioinfo1/Yeast1/C5_C8/C5/chr1/chr1C5.seq
NUCMER
>Chr 9 230208 57
70663 70717 1 56 1 1 0
-11
0
>Chr 93 230208 50
132588 132636 49 1 0 0 0
0
>Chr 112 230208 111
68025 68137 111 1 3 3 0
33
62
0
>Chr 139 230208 112
162041 162152 1 112 4 4 0
0
>Chr 177 230208 94
161054 161146 1 93 5 5 0
0
```

Figure 3.3-1 Sample Delta file

The alignments produced during the comparative assembly process are then filtered and a layout is created from the selected read alignments. The layout is further used to construct the contigs. For our purpose of mutation detection, this delta file plays an important role. Mutations between MA lines and the reference sequence can be detected

using the alignments in this delta file, which is created individually for each assembly. Thus, each sister MA line is assembled against the same reference genome which produces a delta file for each.

3.3.2 Phase I, step 2: Alignment Summary

The MD tool takes this delta file and creates the actual alignments of the reads against the reference sequence.

Reference Position	Reference Base	Bases of aligned reads
1	C	CCCC
2	C	CCCCCC
3	A	AAAAAA
4	C	CCC-CCCC
5	A	AAAAAAA
5.1	-	CC
6	C	CCCC-CCCC
7	C	CCCCCC
7.1	-	T
8	A	AAACAAA
9	C	CCCCCCCC
10	C	CC-CC
11	C	CCACCCC
11.1	-	AAAAAA

Table 3.3-1 Sample Alignment Summary.

Table 3.3-1 shows a sample section from an alignment summary table that is created by the MD tool using the delta file as the input. The table shows the alignment summary for each position in the reference sequence, which is identified in the 'Reference Position' column. The 'Reference Base' column denotes the corresponding base in the reference sequence for that position. The 'Bases of aligned reads' column denotes the bases in the sequence reads that aligned with the reference sequence at that particular position. The number of characters in the column represents the number of reads that aligned with the

reference sequence at that particular reference position. E.g. for the reference position 1, there are 4 'C's in that row, denoting there were 4 read sequences that aligned with the reference and all 4 of them had a 'C' at that position. Also, the insertion positions (e.g. 5.1, 7.1 and 11.1 in the table) denote that the reference sequence did not have the corresponding base, but the base was seen in the read sequences that aligned with the reference sequence. A "-" in the 'Bases of aligned reads' column denotes that the sequence read that aligned with the reference sequence had a deletion at that particular base position. E.g. for the reference base positions 4, 6 and 10, one of the reads that aligned at each of these positions had a base missing in it.

3.3.3 Phase I, step 3: Processing Alignment Summary

The MD output file which gives the alignment summary is further processed to eliminate insertion positions that are seen in the read sequences (Table 3.3-2).

Reference Position	Reference Base	Bases of aligned reads
1	C	CCCC
2	C	CCCCCC
3	A	AAAAAA
4	C	CCC-CCCC
5	A	AAAAAAA
5.1		CC
6	C	CCCC-CCCC
7	C	CCCCCC
7.1		T
8	A	AAACAAAA
9	C	CCCCCCCCC
10	C	CC-CC
11	C	CCACCCC
11.1		AAAAAA

Table 3.3-2 Removing the insertion positions.

Also, the reference positions that are within a 10 base pair window of homopolymers of length 4 or more are eliminated.

Reference Position	Reference Base	Bases of aligned reads
50	T	TTTTTTTT
51	C	CCCCCCC
52	C	CCCCCCCC
53	A	AAAAAAA
54	C	CCC-CCCC
55	A	AAAAA
56	C	CCCCCC
57	C	CCCCCTCC
58	C	CCC-CCCCC
59	C	CCCCCCCC
60	A	AAAA
61	C	CCCC
62	C	CCC
63	T	TTTTTTTT
64	G	GGGGGG
65	A	AAAAAAA
66	G	GGGGGG

Table 3.3-3 Homopolymer region

Reference Position	Reference Base	Bases of aligned reads
50	T	TTTTTTTT
51	C	CCCCCCC
52	C	CCCCCCCC
53	A	AAAAAAA
54	C	CCC-CCCC
55	A	AAAAA
56	C	CCCCCC
57	C	CCCCCTCC
58	C	CCC-CCCCC
59	C	CCCCCCCC
60	A	AAAA
61	C	CCCC
62	C	CCC
63	T	TTTTTTTT
64	G	GGGGGG
65	A	AAAAAAA
66	G	GGGGGG

Table 3.3-4 Eliminating 5 base positions surrounding homopolymer region of 4 or more bases

Table 3.3-3 shows the homopolymer region of length 4 (reference base positions 56 through 59 have the same base “C”). Table 3.3-4 shows the 5 base positions before and after a homopolymer region of 4 or more bases, which are removed from further processing.

We eliminate the 5 base positions surrounding a homopolymer region of 4 or more bases to avoid local alignment problems with read-length errors that can occur at homopolymer regions. The MD output file generated for each strain that is aligned against this reference sequence will eliminate base positions 51 through 64 from further processing, thus eliminating false positive mutations being detected due to sequencing errors in this region. As described earlier, the 454 sequencing technique is unable to identify homopolymers of length more than 4, which results in misalignments in areas with homopolymers.

3.3.4 Phase II, step 1: Combining MD output files

The MD output file generated for each MA line in Phase I is then combined with the output files generated for its sister MA lines. For each MA line (strain), we analyzed its MD output with its sister MA lines to generate a “Combined Output file” in 2 ways:

1. Combine the MD output file with any one of its sister MA lines (Pair-wise Analysis). E.g. the A1 strain was combined with one of the remaining strains: A4, C5 or C8. Analysis was done for the A1-A4, A1-C5 and C1-C8 pair-wise combinations.
2. Combine the MD output file with the remaining sister lines considered as one MA line (All-strain Analysis). E.g. A1 strain was combined with A4, C5 and C8 at the same time. Analysis was done for A1 vs. A4C5C8 treated as a single strain.

For pair-wise as well as all-strain analysis, the combined output file gives the information about the number of reads and their corresponding bases aligning against the reference, for each base position in the reference, for the strains that are analyzed.

Reference Position	Reference Base	Bases of aligned reads in A1
1	A	AAAA
2	T	TTT
3	G	GG
4	T	TT-T

Table 3.3-5 Sample MD output file for strain A1

Reference Position	Reference Base	Bases of aligned reads in A4
1	A	ATA
2	T	TTTT
3	G	G--G
4	T	TTTTTTTTATTATT

Table 3.3-6 Sample MD output file for strain A4

Reference Position	Reference Base	Bases of aligned reads in C5
1	A	TTTT
2	T	TTTTTT
3	G	GGGGCG
4	T	TTTTTT

Table 3.3-7 Sample MD output file for strain C5

Reference Position	Reference Base	Bases of aligned reads in C8
1	A	AAAAA
2	T	TTTGTT
3	G	GGGG
4	T	TTTT

Table 3.3-8 Sample MD output file for strain C8

Table 3.3-5 through Table 3.3-8 show sample lines from MD output files for strains A1, A4, C5 and C8 respectively. Table 3.3-9 shows lines from the 'Combined Output

file' that is created using the MD output files for A1-A4 pair-wise analysis. Similarly, combined output files were created for A1-C5, A1-C8, A4-C5, A4-C8 and C5-C8.

Table 3.3-10 shows the combined output file that is created for an all-strain analysis. A1 is analyzed with the A4, C5 and C8 treated as one single strain. The bases of aligned reads in A4, C5 and C8 are combined and then analyzed with A1. This combined output file is used for A1-A4C5C8 all-strain analysis. Similarly, combined output files were generated for A4-A1C5C8, C5-A1A4C8 and C8-A1A4C5 all-strain analysis.

Reference Position	Reference Base	Bases of aligned reads in A1	Bases of aligned reads in A4
1	A	AAAA	AATA
2	T	TTT	TTTT
3	G	GG	G—G
4	T	TT-T	TTTTTTTTATTTTATT

Table 3.3-9 Sample Combined Output file for pair-wise analysis of strains A1 and A4

Reference Position	Reference Base	Bases of aligned reads in A1	Bases of aligned reads in A4, C5, and C8
1	A	AAAA	AATATTTTAAAAA
2	T	TTT	TTTTTTTTTTTTTGTT
3	G	GG	G—GGGGCGGGGG
4	T	TT-T	TTTTTTTTTATTTTATTTTTTTTTTTTTT

Table 3.3-10 Sample Combined output file for all-strain analysis A1-A4C5C8

3.3.5 Phase II, step 2: Applying coverage restrictions

The combined output file that is generated as shown in section 3.3.4 is further processed to eliminate reference base positions that are covered by fewer than 3 or more than 10 sequence reads (for pair-wise analysis) or more than 30 sequence reads (for all-strain analysis) during the assembly. Coverage of fewer than 3 reads can be considered as insufficient for correctly identifying the base.

We also need to eliminate positions with too much coverage since such positions are most likely part of a region that has replicated, perhaps many times. When this has happened a sequence from one of the replicated regions will align with all of them. Since it is not possible to know where the “correct” alignment is, we cannot determine whether a mutation has occurred in these positions – so they are eliminated from further consideration. For each strain, we had 5x coverage of the entire genome. Hence, we assume that coverage of more than 10 reads for any strain identifies regions that are highly repetitive. If a reference base position has more than 10 reads aligning at that position, it is eliminated from further processing for a pair-wise analysis. For an all-strain analysis, the upper limit for reads is set to 30 rather than 10 since one line is compared against the combined consensus of the 3 MA lines. (If there were 3 MA lines being studied, the all-strain analysis would analyze one strain against the combined consensus of the remaining two strains. In this case, coverage of more than 20 for the combined consensus would be discarded.)

Reference Position	Reference Base	Bases of aligned reads in A1	Bases of aligned reads in A4
1	A	AAAA	AATA
2	T	TTT	TTTT
3	G	GG	G G
4	T	TT-T	TTTTTTTTTATTTTATT

Table 3.3-11 Applying coverage restrictions for pair-wise analysis

Reference Position	Reference Base	Bases of aligned reads in A1	Bases of aligned reads in A4C5C8
1	A	AAAA	AATATTTTAAAAA
2	T	TTT	TTTTTTTTTTTTGTT
3	G	GG	G GGGGGGGGGGG
4	T	TT-T	TTTTTTTTTATTTTATTTT

Table 3.3-12 Applying coverage restrictions for all-strain analysis

Table 3.3-11 shows that base position 3 is eliminated from further processing since the coverage at the position for strain A1 is less than 3. Also, base position 4 is eliminated since coverage at that position for strain A4 is more than 10.

Table 3.3-12 shows that for all-strain analysis for A1-A4C5C8, base position 3 is eliminated from further processing since the coverage at that position for A1 is less than 3. Base position 4 is not eliminated because the combined coverage of A4, C5 and C8 at that position is less than 30.

3.3.6 Phase II, step3: Creation of analysis tables

The majority of the type of base in the aligned reads of an MA line for any particular base position decides the consensus. For example if 5 reads of an MA line align at a particular base position of the reference, and 3 out of them have the base 'A' at that position, while the other 2 reads have a different base, then the consensus for that base position will be 'A'.

The processed combined output file generated as described in section 3.3.5 is used to generate additional information about the consensus sequence for each strain that is analyzed. Table 3.3-13 shows the two columns 'A1 consensus' and 'A4 consensus' that are generated using the data in the 'Bases of aligned reads' column for each strain. The consensus produced by each MA line, for each base position in the reference is the key information used for mutation detection. Note that base positions 3 and 4 are eliminated as explained in Table 3.3-11. This processed combined output file is further used to generate 4 different analysis result tables: Errors/Matches Matrices, Reference Difference Positions, Mutations, and Putations. The creation of the analysis tables is explained in detail in the following sections.

Reference Position	Reference Base	Bases of aligned reads in A1	Bases of aligned reads in A4	A1 consensus	A4 consensus
1	A	AAAA	AATA	A	A
2	T	TTT	TTTT	T	T

Table 3.3-13 Consensus creation

3.3.7 Errors/Matches

The 4 MA lines that we used were all haploid. This essentially means that all the copies of the same 454 sequence for a particular line should be identical. If they are not, then we can conclude that it is because of sequencing errors or because of misalignments in the assembly process in highly repetitive regions. Errors in base calls can affect the mutation detection result.

The consensus failure rate is a combination of the false positive rate due to sequencing errors and the fact that consensus cannot be reached at all due to lack of majority base type. Thus, the false positive rate arising due to sequencing errors contributes to the consensus failure rate. So, it is imperative to find the sequencing error rate while identifying the mutations.

We determined the 454 sequencing error rate by creating Error and Match matrices. The error and match information is gathered based upon the number of nucleotides that align to the same reference base position for the two genomes. For a particular base position in the reference, the number of errors is calculated as the number of differences between the bases of the reads spanning that position as compared to the consensus produced by the reads. The matches are the number of bases that are the same as the consensus. The number of “errors” found at a given base position, along with the coverage of reads at that base position is used to generate what we call the “Error Matrix”. Similarly, the number of “matches” found at a given base position, along with

the coverage of reads at that base position is used to generate the “Match Matrix”. These Error and Match matrices are used to detect the sequencing error rate of the 454 reads. For calculating the errors and matches at a given base position, the reference base can be anything. Matches and errors are calculated depending upon the consensus sequence for the two strains.

Table 3.3-14 shows the computation of match and error values for three different positions. ‘N’ signifies that the corresponding base can be either ‘A’, ‘T’, ‘G’ or ‘C’. At position 100 in the reference, 3 matches for A1 and 3 matches for A4 are found and hence 6 matches are counted for position 100. Similarly, 6 matches are counted for position 105 in the reference. For position 300, 2 matches for A1 and 2 matches for A4 account for 4 matches and 1 error from each strain accounts for 2 errors in all.

Position	Reference	A1	A4	A1 consensus	A4 consensus	Remark
100	N	AAA	AAA	A	A	6 matches
105	N	AAA	TTT	A	T	6 matches
300	N	AAT	TTA	A	T	4 matches, 2 errors

Table 3.3-14 Calculation of Errors and Matches for a base position

We now use an example for 10 base positions and show how the errors and matches ultimately define the Error Matrix and the Match Matrix respectively for an analysis of 2 MA lines. Note that this is just an example and not the actual data that we have for the lines A1 and A4. In Table 3.3-15, the reference base is not shown since it is not required to generate the Match and Error matrices. It includes the reference position, bases of A1 and A4 reads which span that position and the errors and matches calculated for those positions using the method shown in Table 3.3-14.

Position	A1	A4	A1 consensus	A4 consensus	Remark
101	AAA	AAA	A	A	6 matches
102	AAC	TTTA	A	T	5 matches, 2 errors
103	ATT	TTATT	T	T	6 matches, 2 errors
104	CCC	TTT	C	T	6 matches
105	TTC	TTTA	T	T	5 matches, 2 errors
106	GGGG	GGA	G	G	6 matches, 1 error
107	GAAAA	CCC	A	C	7 matches, 1 error
108	GGGG	GGC	G	G	6 matches, 1 error
109	CCCT	AAAT	C	A	6 matches, 2 errors
110	TTTTT	ACA	T	A	7 matches, 1 error

Table 3.3-15 Sample data for 10 reference positions

We want to determine if there is any relationship between error and match rates and coverage for the two genomes. Consequently, we want to compute summary information based on position coverage. In other words, we would like to summarize the error and match information for each possible combination of coverage of a given position for the two genomes. Since we allow coverage to range from 3 to 10 for pair-wise analysis, we need one 8x8 matrix to represent a composite summary of the relationship between coverage at a reference position and the error information and another 8x8 matrix to represent the composite match information. We call the 8x8 matrix that comprises the error information as the “Error Matrix” and we call the 8x8 matrix that comprises the match information as the “Match Matrix”. For an 8x8 matrix, the matrix indices are between (0, 0) to (7, 7). Matrix position (0, 0) represents coverage of 3 for both the MA lines while the matrix position (7, 7) represents coverage of 10 for both the MA lines. Similarly position (0, 1) represents coverage of 3 for the first strain (A1) and 4 for the second strain (A4); and so forth. The entire matrix represents all the cases where the

reference base position is covered by a variable number of reads spanning it for the 2 MA lines. For example, in our sample Table 3.3-15, the reference base position 102 is spanned by 3 reads of strain 'A1' and 4 reads of strain 'A4' during the alignment. Thus the matrix position where the errors and matches would be counted for this particular base position would be (0, 1). There are 5 matches and 2 errors at this base position. The 5 matches will be counted at the matrix position (0, 1) in the 'Match Matrix' while the 2 errors will be counted at the matrix position (0, 1) in the 'Error Matrix'. Thus if the first MA line has a coverage of 'x' and the second MA line has a coverage of 'y' for a particular base position, then the errors and matches for that position would be summarized at the matrix position (x-3, y-3) in the Error Matrix and the Match Matrix respectively.

Position	A1 coverage (x)	A4 coverage (y)	Matrix Position (x-3, y-3)
101	3	3	(0, 0)
102	3	4	(0, 1)
103	3	5	(0, 2)
104	3	3	(0, 0)
105	3	4	(0, 1)
106	4	3	(1, 0)
107	5	3	(2, 0)
108	4	3	(1, 0)
109	4	4	(1, 1)
110	5	3	(2, 0)

Table 3.3-16 Matrix position calculation.

Table 3.3-16 shows how the matrix position is calculated for our sample data in Table 3.3-15. The errors for base position 101 is summarized in the Error Matrix at the position (2, 0) while the matches for this position is summarized in the Match Matrix at the position (2, 0). The errors mapping to a particular matrix position in the Error Matrix are added together. Similarly, the matches mapping to the same matrix position in the Match

Matrix are added together. Based on Table 3.3-16, we know that the errors for base positions 101 and 104 would be mapping to the same position in the Error Matrix, which is position (0, 0). Their matches also map to the same position in the Match Matrix. Hence in the Error Matrix, at position (0, 0), the errors for base positions 101 and 104 are added together. Similarly, their matches are added together at position (0, 0) of the Match Matrix.

Matrix position (x, y). y → x ↓	0	1	2	3	4	5	6	7
0	0 (101 + 104)	4 (102 + 105)	2 (103)	-	-	-	-	-
1	2 (106 + 108)	2 (109)	-	-	-	-	-	-
2	2 (107 + 110)	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-

Table 3.3-17 Error Matrix.

Table 3.3-17 shows the 'Error Matrix' for the sample data in Table 3.3-15. The reference positions in the brackets indicate that the errors (calculated in Table 3.3-15) for these positions are added together for that particular matrix position. The '-' indicates that there was no data available for these cells in the Error Matrix. This also means that there was no A1 or A4 coverage for any of the reference positions which could be mapped to those cells in the Error Matrix.

Table 3.3-18 shows the 'Match Matrix' for the sample data in Table 3.3-15. Similar to the Error Matrix, the matches for a reference position as calculated in Table 3.3-15 are added to this Match Matrix using the matrix position in Table 3.3-16. The '-' means that there was no A1 or A4 coverage that could be mapped to those cells in the Match Matrix and hence are kept empty. Similar to the Error Matrix, this is an 8x8 matrix with 'x' and 'y' indices ranging from '0 to '7'.

Matrix position (x, y). y → x ↓	0	1	2	3	4	5	6	7
0	12 (101 + 104)	10 (102 +105)	6 (103)	-	-	-	-	-
1	12 (106 + 108)	6 (109)	-	-	-	-	-	-
2	14 (107 + 110)	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-

Table 3.3-18 Match Matrix.

Using the Error and Match Matrix, the base calling error at a particular matrix position was calculated as follows:

$$BCE_{ij} = E_{ij} / (E_{ij} + M_{ij})$$

where BCE_{ij} is the Base Calling Error at position (i, j)

E_{ij} is the value of the Error Matrix at position (i, j)

M_{ij} is the value of the Match Matrix at position (i, j)

Thus, for matrix position (0, 1), the base calling error rate is $4 / (4 + 10) = 0.286$. For all the MA lines, after computing the Error and Match Matrix, we calculated the base calling error rate in the manner described above for each matrix position. Then we calculated the average of that error and found out that the total base calling error rate was 0.13%. This error rate was approximately 4 times less than what we had expected from the previous studies where the base substitution sequencing error rate of 0.5% is assumed [7]. The low base calling error rate meant that the quality of sequences we worked with was really good. This also yielded 10-10,000 times (depending upon the coverage) reduction in the false positive rate.

3.3.8 Reference Difference Positions

For a particular base position in the reference, if all the reads spanning this position have the same base in both MA lines, but this base is different from the reference base, then the base position is counted as a *reference difference* position. An example of this situation is shown in Table 3.3-19. These positions indicate where the same base mutations have occurred with respect to the reference sequence, presumably in a progenitor of the MA lines.

Position	Reference	A1	A4	A1 consensus	A4 consensus
1020	A	TTT	TTT	T	T

Table 3.3-19 Reference Difference positions

3.3.9 Mutations

Irrespective of the reference base, if the consensus base for the MA lines for a particular reference position is different, then this base position constitutes a mutation between the two lines. The consensus should be such that all the bases constituting the

consensus are the same and thus it is a unanimous consensus. These positions represent confirmed mutations between the 2 MA lines under observation.

Table 3.3-20 shows two variations of reference difference positions. In the first case, a definite mutation from A to T can be seen along the A4 line with respect to the reference. This can be detected only by comparing the consensus of A1 and A4 to the reference base. In the second case, it cannot be known for sure exactly where the mutation between the reference and the 2 strains occurred during the evolutionary process.

Position	Reference	A1	A4	A1 consensus	A4 consensus
1040	A	AAA	TTT	A	T
1060	T	AAA	CCC	A	C

Table 3.3-20 Mutations

3.3.10 Putations (Putative Mutations)

Irrespective of the reference base, if the 2 lines have a different consensus, and the consensus consists of some bases that are not the same as the consensus, then such reference base positions constitute *putations* between the 2 lines. Putations can be considered as mutations that are not confirmed, since not all the reads spanning the base position have the same base. This can be attributed to the possibility that this read aligned at more than one place in the reference because of its occurrence in a repetitive region. This can also be attributed to a 454 sequencing error.

Position	Reference	A1	A4	A1 consensus	A4 consensus
1080	N	AAT	TTT	A	T
1090	N	AATTT	AAA	T	A

Table 3.3-21 Putations

In Table 3.3-21 'N' signifies that the corresponding base can either 'A', 'T', 'G' or 'C'.

3.4 Implementation

The various scripts needed to implement the mutation detection approach were first implemented in Perl. The results from the scripts were analyzed and it was confirmed that we were able to find many more mutations by comparing the assembly results of different strains as compared to the number of mutations reported by 454 after assembling the data using their Newbler Assembler. For chromosomes in the range of 200,000 base pairs, it took approximately 5 hours to produce the output files. For longer chromosomes (1,400,000 base pairs long), it took 12-13 hours. Although useful results were obtained, the process was time consuming.

To improve the performance, the entire process was re-implemented in Java. The Perl script used the *show-aligns* utility provided by the MUMmer package to get the alignment information of an input sequence to the reference sequence. Processing the output of this utility consumed a lot of time. To avoid this, the Java implementation reads the reference sequence separately and the alignment information is extracted from the output of the delta file that is created as one of the outputs of the assembly process. Due to these changes, the Java implementation shows significant performance improvement over the Perl implementation. For smaller chromosomes (~200,000 base pairs length), it now takes around 1 minute to get the output and for larger chromosomes (~1,400,000 base pairs long), it takes 2-3 minutes to get the output.

3.5 Results

We have used the MD tool with 4 yeast MA lines - A1, A4, C5 and C8. These lines were comparatively assembled against *S. cerevisiae* as the reference sequence. The output from each assembly process was then provided as input to the MD tool. During

Phase I of the MD tool approach, insertion positions and homopolymer regions were eliminated from further processing.

Initially, each chromosome in the yeast genome was assembled separately against the reference chromosome sequence (chromosome assembly). But, once the performance was improved by the Java implementation, the entire genome of a strain was assembled against the entire genome of the reference sequence (genomic assembly). Table 3.5-1 compares the results of these two types of assemblies and the number of positions assayed in each one.

Genomic Assembly Vs. Chromosome Assembly						
Strain	Assembled positions with respect to reference sequence		After removing insertions		After eliminating homopolymer regions of 4 or more.	
	Genome	Chromosome	Genome	Chromosome	Genome	Chromosome
A1	11,810,882	11,847,100	11,133,368	11,487,142	8,169,073	8,445,134
A4	12,020,356	12,305,578	11,328,367	11,729,731	8,299,926	8,615,613
C5	10,505,708	11,023,547	10,337,691	10,794,238	7,596,643	7,922,054
C8	10,766,874	11,293,467	10,557,340	11,014,515	7,761,680	8,089,638

Table 3.5-1 Assembled positions in Genomic Assembly Vs. Chromosome Assembly

After the comparative assembly, insertions within the MA lines were ignored. Also, by neglecting the regions around the homopolymers of length more than 4, any false positive mutations detected due to sequencing errors were avoided.

We found more mutations with a chromosome assembly because many reads that aligned with different chromosomes were also included in this assembly. These reads may not have had exact alignments since they aligned perfectly to other chromosomes. This resulted in more false positive mutations being detected. Because of the whole genome assembly, we got better alignments and better assembly results since the reads

were no longer ambiguously aligned to the reference sequence. This reduced the number of alignment errors and therefore also the number of putations.

After the individual strains were comparatively assembled with the reference sequence, during Phase II, the assembly results for individual strains were compared with one another and the mutation analysis was carried out on 2 strains at a time (pair-wise analysis) or a combined analysis using all 4 strains (all-strain analysis). During the all-strain analysis, one strain was compared against a combination of the other three strains to find the mutations. The error and match matrices were created for both types of analysis. These matrices were generated based upon the alignment coverage for each strain for the same reference position. The goal was to see if there was any difference in the base calling, based on the relative coverage for each strain. Our analysis showed that there was no significant difference.

Chr#	Analysis type	Positions assayed	Positions with 3-10 or 3-30 coverage
1	A1-A4	226,746	103,651
1	C5-A1	167,111	71,751
1	C5-A4	167,591	81,620
1	C8-A1	167,080	61,659
1	C8-A4	167,682	68,016
1	C5-C8	162,522	66,115
1	A1-A4C5C8	230,208	96,340
1	A4-A1C5C8	230,208	110,725
1	C5-A1A4C8	230,208	129,200
1	C8-A1A4C5	230,208	86,826

Table 3.5-2 Pair-wise strain analysis Vs. All-strain analysis

Table 3.5-2 shows that more reference positions are assayed with the all-strain analysis as compared to the pair-wise analysis. A reference position that is not assayed in a pair wise assembly can be assayed in other assemblies thus increasing the total number of

reference positions assayed. Also, restricting coverage from 3-10 for pair-wise and 3-30 for all-strain analysis avoids assaying positions in repeat regions. Since all-strain analysis was better than pair-wise, the mutation analysis was further carried out using the results of all-strain analysis.

Table 3.5-3 compares the genomic and chromosome assemblies with respect to the analysis tables (reference difference, mutations and putations) generated for each type of assembly. The table shows that the number of putations decreased for the genomic assembly as compared to the chromosome assembly. By comparing strains with each other and also with the reference sequence, exact mutations occurring within a particular MA line are detected. This would not have been possible if a single strain had been compared with only the reference sequence.

Genomic Assembly vs. Chromosome Assembly						
Analysis	Reference Differences		Mutations		Putations	
	Genome	Chromosome	Genome	Chromosome	Genome	Chromosome
A1-A4C5C8	196	159	9	14	1104	1367
A4-A1C5C8	244	214	13	9	1084	1544
C5-A1A4C8	141	157	14	59	933	2394
C8-A1A4C5	132	149	16	47	932	2341

Table 3.5-3 Analysis tables for Genomic Assembly vs. Chromosome Assembly

Table 3.5-4 shows the entire list of mutations in the Yeast MA lines that the MD tool was able to find.

Yeast MA Line Mutations
Mutation Table as of 6 April 2006

Location	Change	Context	Line(s)
Substitutions			
Chr. 2-125366**	Tv (G→C)	IGTGTCCAAG→TGTGTGCAAG	A4*
Chr. 2-536163	Tv (G→C)	AGAAGAAACG→AGAACAAACG	C8
Chr. 2-696533	Tv (G→T)	AGAACGTCTT→AGAACTTCTT	A4, A5
Chr. 3-54214	Ts (T→C)	CCTTACAAA→CCTTCACAAA	C8
Chr. 4-1148647	Tv (G→T)	GGATGCTAAG→GGATTCTAAG	C5
Chr 5-351995	Ts (A→G)	TCCTGAGCAG→TCCTGGGCAG	A1
Chr. 7-67430	Ts (G→A)	GGCGGATTTTC→GGCAGATTTTC	C5
Chr 7-561788	Tv (G→C)	AATCGTGCAC→AATCCTGCAC	A1*
Chr. 7-625107	Tv (A→C)	CTTGAAGTGC→CTTGCAGTGC	A4
Chr. 7-804473	Tv (A→C)	TTTGAATTGG→TTTGCATTGG	C8
Chr. 8-231499	Tv (A→T)	TGGATATTGT→TGGATTTTGT	A4*
Chr. 8-262187	Ts (C→T)	GAATAGAACT→GAATAGAATT	A4, A5
Chr. 9-154205	Tv (A→C)	TTATCGAAGG→TTATCGAAGG	C8
Chr. 9-380265	Tv (G→T)	GAGAGTCAGCG→GAGATTTCAGCG	C5
Chr 10-33149	Ts (G→A)	GGAAGGATAT→GGAAGAATAT	A1, A2
Chr. 11-239813	Ts (G→A)	ACTAGATTGA→ACTAAATTGA	C8
Chr. 11-605009	Ts (C→T)	TCATCATCCT→TCATTATCCT	C5
Chr. 12-277642	Ts (T→C)	CATTGTCTC→CATTGCTCTC	A1
Chr. 12-617519	Tv (G→C)	GGGAAGGGTT→GGGAACGGTT	A4,A5
Chr. 12-716670	Tv (G→T)	CTATAGAATT→CTATATAATT	A1, A2
Chr. 13-213440	Tv (G→C)	CTGGCCGATC→CTGGCCGATC	A1, A2
Chr. 13-503024	Tv (C→A)	CGAACCGTAA→CGAACAGTAA	A4, A5
Chr. 13-560365	Tv (A→T)	AAACCATAAT→AAACCATTAT	A2
Chr. 13-824994	Tv (C→A)	CCGTCACTCA→CCGTAACTCA	C8
Chr 13-913509	Tv (C→A)	CCTGTGACC→CCTGTAGACC	A4
Chr. 14-688148	"inversion" #	IGCCGAAGGCA→TGCCTTCGGCA	C5
Chr. 15-541599	Ts (T→C)	AATTTCTTAC→AATTCCTTAC	C5
Chr. 15-679548	Ts (G→A)	AACTGCTAAA→AACTACTAAA	C8
Chr. 15-986649	Tv (C→A)	GCTGCGTCGC→GCTGAGTCGC	C8
Chr 16 331354	Ts (G→A)	CAGTGAGGAT→CAGTAAGGAT	C8
Chr 16 804029	Tv (G→T)	AGGTGCCAGA→AGGTGCCAGA	C8
Chr 16 834238	Ts (G→A)	TTATGTGAAA→TTATATGAAA	C8
Indels			
Chr. 4-117354	Deletion of G	TCTGGGGACT→TCTGGG-ACT	A1, A2

*sequencing remaining lines

**Mutation is heterozygous ... possible gene duplication??

#aa change E to F in ABZ1 (aminodeoxychorismate synthase)

<http://db.yeastgenome.org/cgi-bin/locus.pl?locus=308023RW>

Mutations remaining to be confirmed

Chr 4 38790 C to T and 38791 T to G

Chr 16 575685 G to C REDO WITH FORWARD PRIMER

Chr 6 182779 A to T

Chr12 490683 T to G

Chr 10 193580 A to G

Table 3.5-4 Yeast MA line mutations

3.5.1 Base Substitutions

With the help of the mutation detector, we were able to find 33 base substitutions in an average of 4.99×10^6 nucleotide sites that fulfilled our criteria for coverage and distance from a homopolymer region. These mutations were later verified by direct sequencing. This yielded an overall base substitution rate of 0.33×10^{-9} per site per cell division.

Along with base substitution, we were also able to find one single base pair deletion and a 3 base pair inversion.

3.5.2 Reference Difference Positions

Using the method defined in section 3.3.8 for the reference difference positions, we found 323 base positions with consistent consensus sequence across the 4 MA lines that differed from the base in the reference sequence. These are possibly the sequencing errors in the reference sequence or mutations arising in the progenitor of the MA lines before the start of the experiment.

3.5.3 Heterozygous Sites and Mutations

There was evidence found that the MA lines had become diploid in nature spontaneously in the early stages of the experiment. This meant that we would have failed to find mutations at heterozygous sites with our approach of using 'consensus sequence'. We attributed the reason for any base in the reads spanning the same base position of the reference being different from the consensus to be caused by a sequencing error. But, if the lines had been heterozygous, then such base differences would have been in fact a result of heterozygosity.

For each point mutation that we found using our Mutation Detector, we sequenced DNA obtained from frozen cells stored at 5-30 day interval throughout the period of the experiment. We found that all the mutations detected were homozygous at the time of the first detection, except for 1. We also searched the 454 sequence data for potential heterozygotes. For this, we focused on base positions which were spanned by at least 2 reads that had the same base as the reference and at least 2 reads that were different from

the base in the reference sequence. There were 202 potential heterozygous mutations that we found across all 4 MA lines. For each such mutation, we took the surrounding DNA 28 base pairs up and down stream and BLASTed them against the reference sequence. This segment analysis revealed that except for ten, all the other potential mutations were a result of 454 assembly artifacts involving nearly identical paralogous sequences. Out of these ten, eight were false positives; one was a heterozygote that was already determined by initial analysis and only one novel mutation was detected.

This analysis shows that relative to the total length of the experiment, the transient phase of heterozygous mutation was small enough and therefore any failure in determining heterozygous mutations would not have affected our mutation rate detection.

CHAPTER 4

MULTIPLE ASSEMBLY VIEWER WITH ENHANCED

NAVIGATION (MAVEN)

4.1 Introduction

The MAVEN genome assembly visualization tool supports interactive visualization of sequence assembly and alignment results for multiple genomes at the same time. The tool is part of an interactive environment to support research in the rapidly growing field of **comparative genomics**. In this context it is especially valuable to compare how many closely related species assemble to each other or to a common ancestor. Existing **assembly visualization** tools do not provide simultaneous views of multiple contigs, let alone entire genomes. Existing visualization tools for **comparing genomes** do not incorporate assembly information, but instead focus on showing relative locations of conserved regions. MAVEN is able to show simultaneous views of *de novo* assemblies, comparative assemblies, and alignments. The combination of both assembly and alignment results into a single visualization has already helped to provide better insight into these two techniques and how they interact in a comparative genomics environment. A major motivation for the development of MAVEN has come from our efforts to

assemble sequence data generated by 454 Life Sciences using their PicoTiterPlate™ Device. This process generates sequence data that is approximately 1/10 the size of traditional sequence data; existing assembly software does not handle this data very effectively. MAVEN has helped us to identify more quickly where and why current assembly programs fail with this data.

4.2 Motivation

We felt particularly constrained by the existing visualization software that provided us with only very limited tools for exploring the assembly results in a way that would enable us to understand why the assembly program was not successfully assembling sections of the genome. Existing assembly viewers can only display the results of one assembly process at a time and can only show the composition of a single assembled sequence (called a *contig*) at a time. We wanted to be able to simultaneously view multiple contigs and multiple assemblies so that we could compare the effects of different parameters and different assemblers acting on the same input.

This experience led to the design and implementation of MAVEN: Multiple Assembly Viewer with Enhanced Navigation. MAVEN aims to provide support for exploring and comparing results from multiple assembly and alignment steps and will serve as a fundamental component of a comprehensive environment for assembling multiple genomes simultaneously using an iterative, user-guided assembly process. The sections below provide a brief review of an existing typical visualization tool, BankViewer, and a summary of the goals and features of MAVEN along with some examples.

4.3 Existing Tool – BankViewer

The assembly viewer called BankViewer (<http://amos.sourceforge.net/docs/viewer/>) provided by the AMOS package [10], enables users to view all the information related to the assembly process in a single interactive tool. BankViewer is able to display the results of *de novo* as well as comparative assemblies.

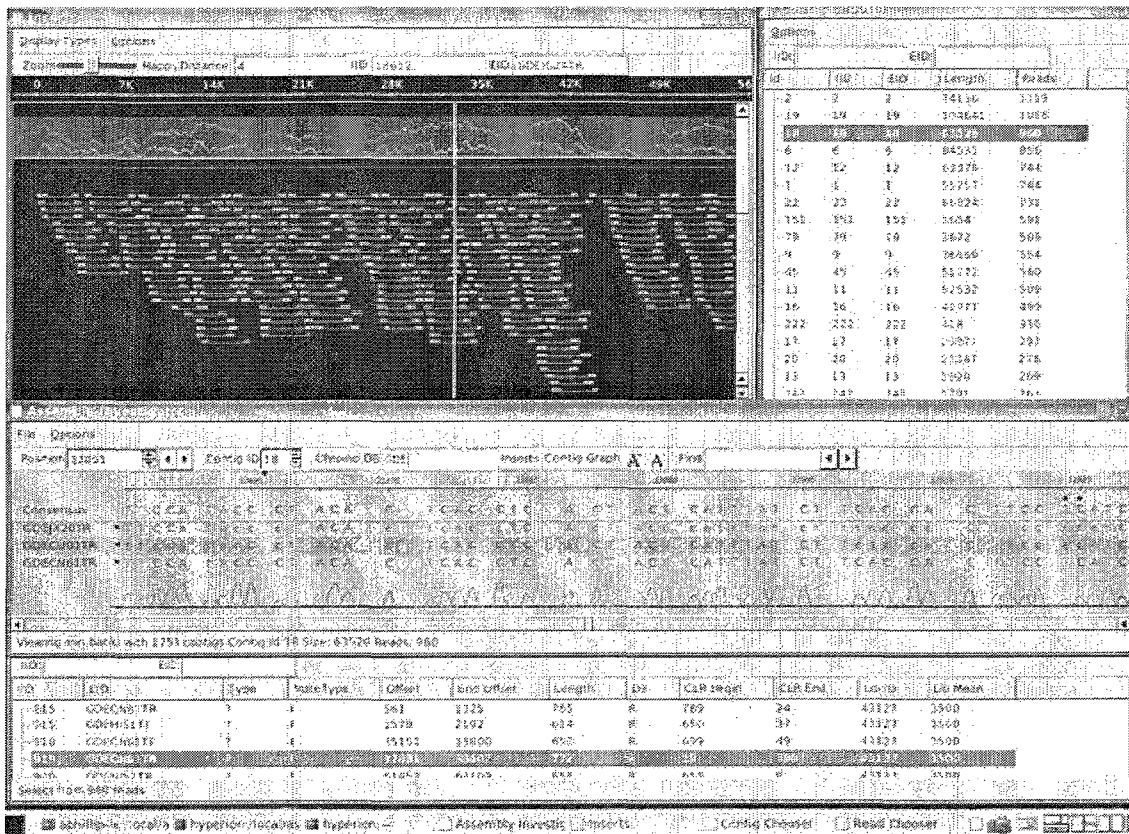


Figure 4.3-1 BankViewer showing one (blue) contig and many (green) reads.

BankViewer displays the contigs of a *de novo* assembly in different windows. The user has to select the contig of interest and only this particular contig is displayed in the main window; it is also able to display the results of only a single assembly process at a time.

Figure 4.3-1 BankViewer showing one (blue) contig and many (green) reads. shows the results of a comparative assembly as displayed by BankViewer. The nucleotide view is shown in the middle; sequence details are shown on the bottom; and contig data on the

right. The upper portion of the figure shows the alignment information. The reference sequence (white) is at the very top. Below the reference sequence are 2 coverage graphs. Below the graphs are the blue contigs aligned with the reference. (Only 1 contig is visible in this view.) The input read sequences (green) are shown below the contigs where they align with the contig. Other portions of the figure show sequence read detail (on the right), contig detail information (on the bottom) and a nucleotide view of the selected contig, including the read sequences that align to it and the consensus sequence.

Limitations of BankViewer

The current functional goals for MAVEN are primarily influenced by BankViewer. MAVEN aims at improving the usability of BankViewer by eliminating two of its major limitations:

For a *de novo* assembly, BankViewer displays only a single contig in a window.

BankViewer is able to display the results from only one assembly process at a time.

4.4 MAVEN as an Assembly Test and Tuning Tool

Eventually, we envision MAVEN supporting a multigenome iterative assembly process. Currently, the ability to simultaneously show results from multiple assemblies and alignments makes MAVEN useful as an assembly testing and tuning tool. MAVEN functionality helps users to better understand the assembly process and to more effectively explore the effects of various assembly parameter choices.

Our initial goal was to use the TIGR assembler to assemble 454 data. But, our initial *de novo* assembly efforts were disappointing; the assembly produced more contigs (of shorter length) than we had hoped. We subsequently used the AMOS comparative assembler to assemble these contigs against a reference genome of a very closely related

species. This approach enabled us to validate the basic assembly results of the *de novo* assembler, to see where it had failed to join sequences that belonged together, and to gain insight into the shortfalls of the assembly algorithm.

MAVEN aids in such analyses because:

1. The *de novo* assembly for the same set of sequence reads could be performed using many different parameter settings
2. The *de novo* assembly outputs were input to the comparative assembler using the same reference sequence.

These comparative assembly results can be viewed using MAVEN to get an idea of read coverage and sequence alignments for each of the assemblies. Depending upon these comparisons, the best assemblies were identified and we were more easily able to converge on a set of parameters for more effective *de novo* assembly of this kind of data. With the ability to view multiple comparative assemblies in the same window, it is much easier to see the contig alignment of two different genomes against a common reference. In the future, we plan to use the matching portions between 2 assemblies to create global alignments between the assembled genomes. The global alignment will then represent a potentially conserved region inherited from the ancestral genome of the assembled genomes.

4.5 Results

Currently, MAVEN is able to display the results of *de novo* assemblies done by the TIGR assembler [9, 12] and comparative assemblies done by AMOS [10]. It can display multiple assemblies in the same window. It has a zoom adjustment that can be used to change the view resolution.

MAVEN is also able to display the alignments of the sequences against a reference genome. The alignments are produced as a result of the 'nucmer' alignment tool which is part of the MUMmer package [1], available online at <http://mummer.sourceforge.net/>

The AMOS comparative assembler aligns input sequences against the reference sequence; those that align well become the input for the local assembly of contigs. Not all aligned sequences, however, contribute to contigs; these are discarded by AMOS and are not included in its final output. By including both the aligned sequences and the sequences that make up contigs in a single window, MAVEN is able to give a better idea as to why some of the aligned sequences are not included in the assembly.

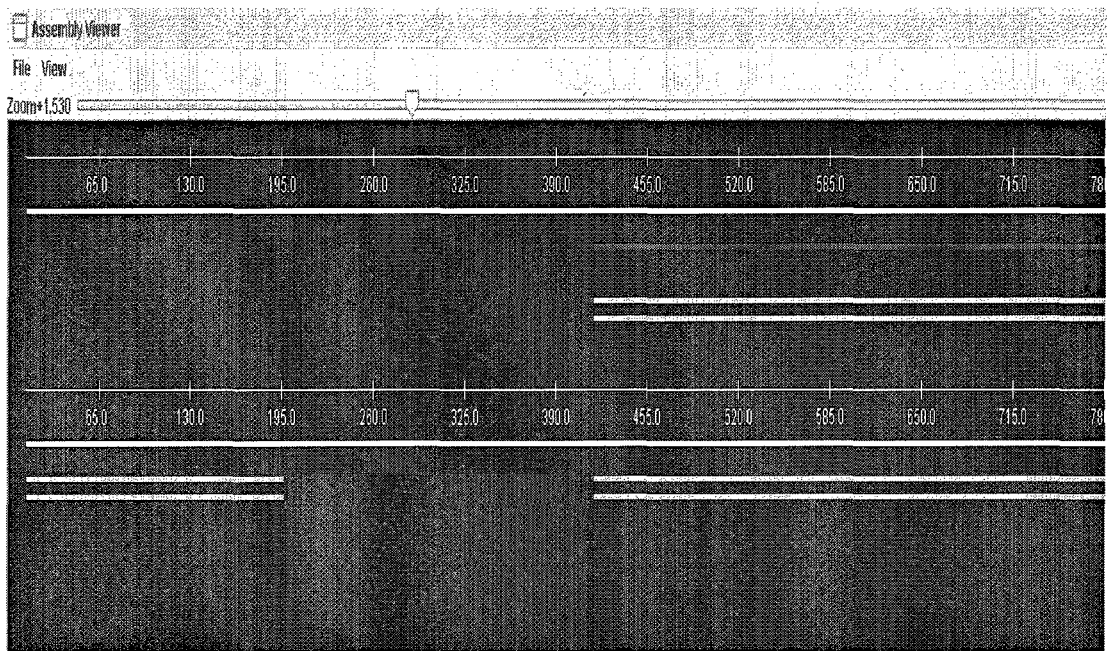


Figure 4.5-1 Reads discarded from contig formation step in the assembly.

Figure 4.5-1 displays one such scenario where the alignment is produced, but during the contig formation, the reads are discarded by the AMOS assembler. The top portion of Figure 4.5-1 displays the result of comparative assembly for a set of 4 test reads against a sample reference sequence. Out of 4 reads, 2 reads aligned from base position 1 to 65

and 125 to 195 with respect to the reference sequence. Thus, these 2 reads had a gap of 60 base pairs as compared to the reference sequence. The other 2 reads matched completely with the reference sequence from base position 421 to 900. These 4 reads were comparatively assembled using AMOScmp, and the alignment produced for the 4 reads is shown in the bottom part of Figure 4.5-1. The first 2 reads produced an alignment from base position 1 to 195 with respect to the reference sequence. AMOScmp was able to detect the gap of 60 base pairs in the reads. Because of this gap, the reads were discarded from the assembly since the percent identity match with the reference for these reads decreased. MAVEN is able to display the comparative assembly output along with the read alignments and so the case where reads are discarded from the assembly even when they align with the reference sequence, can be detected immediately.

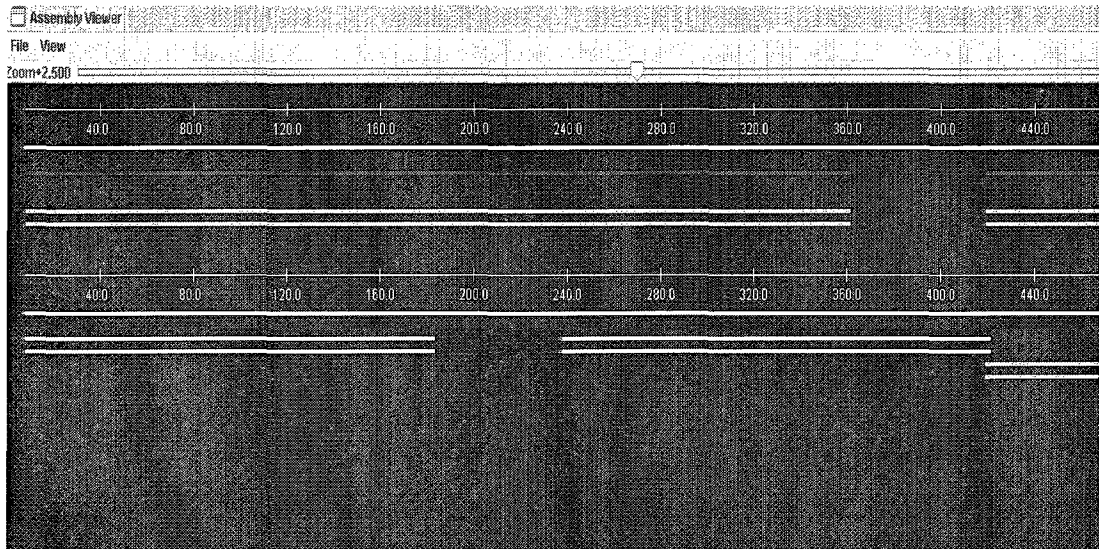


Figure 4.5-2 Parts of a read aligning at different positions in the reference sequence

Figure 4.5-2 again displays a comparative assembly result and the alignment result for 4 test reads against a sample reference sequence. Two of the four reads are of particular interest. The alignment produced for these 2 reads shows 2 separate alignments for 2 separate parts of the read. Both these reads are identical and are 360 base pairs long. The

first 180 bases of the reads align from base position 1 to 180 in the reference. The next 180 bases of the reads align from base position 240 to 420 in the reference sequence. A single alignment with a gap is not produced. Due to this, the assembly accepts these 2 reads for the further contig formation step. But in the actual layout the contig aligns from base position 1 to 360 with respect to the reference. MAVEN is able to display the assembly as well as the alignment results for such reads making it evident why the reads were included in the contig formation step.

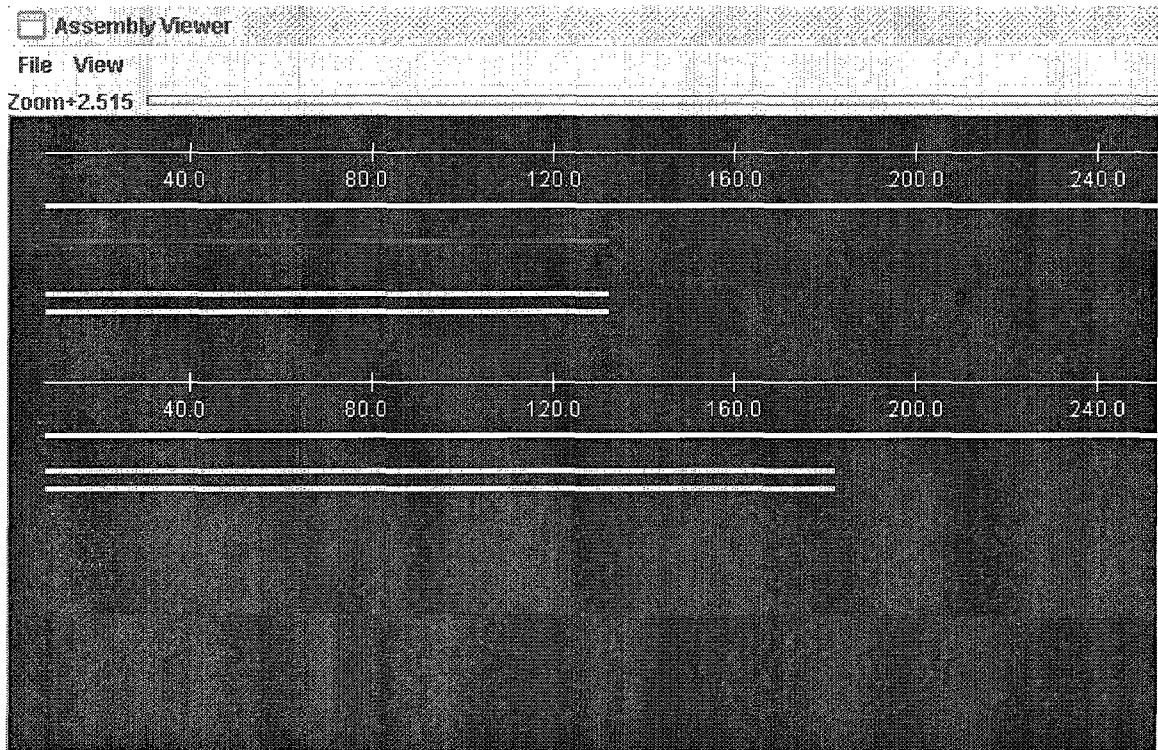


Figure 4.5-3 Reads with gaps accepted in contig formation.

Figure 4.5-3 displays yet another scenario where the 2 reads produce a single alignment with a gap of 50 base pairs. They align from base position 1 to 180 with respect to the reference. Even though this case is similar to the one shown in Figure 4.5-2, these two reads are selected for the contig formation step and they form a contig that aligns from base position 1 to 130 with respect to the reference sequence. These 3

cases (Figure 4.5-1 through Figure 4.5-3) represent discrepancies between the read alignment and the actual contig formation and were easily identified with MAVEN because of its ability to display multiple assemblies at a time.

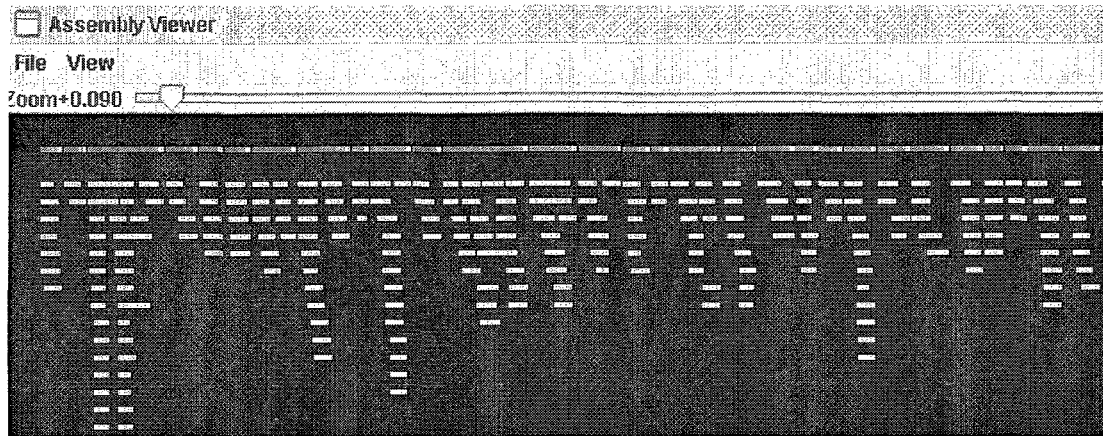


Figure 4.5-4 A view of *de-novo* assembly results.

Figure 4.5-4 displays the *de novo* assembly results. The red lines indicate the contigs and the cyan lines indicate the sequence reads that made up the particular contig. Since there is no reference sequence for the *de novo* assembly, the placement of contigs next to each other is arbitrary. The pink at the end of a contig helps distinguish the contigs when the assembly view is zoomed out and the contigs are displayed very close to each other.

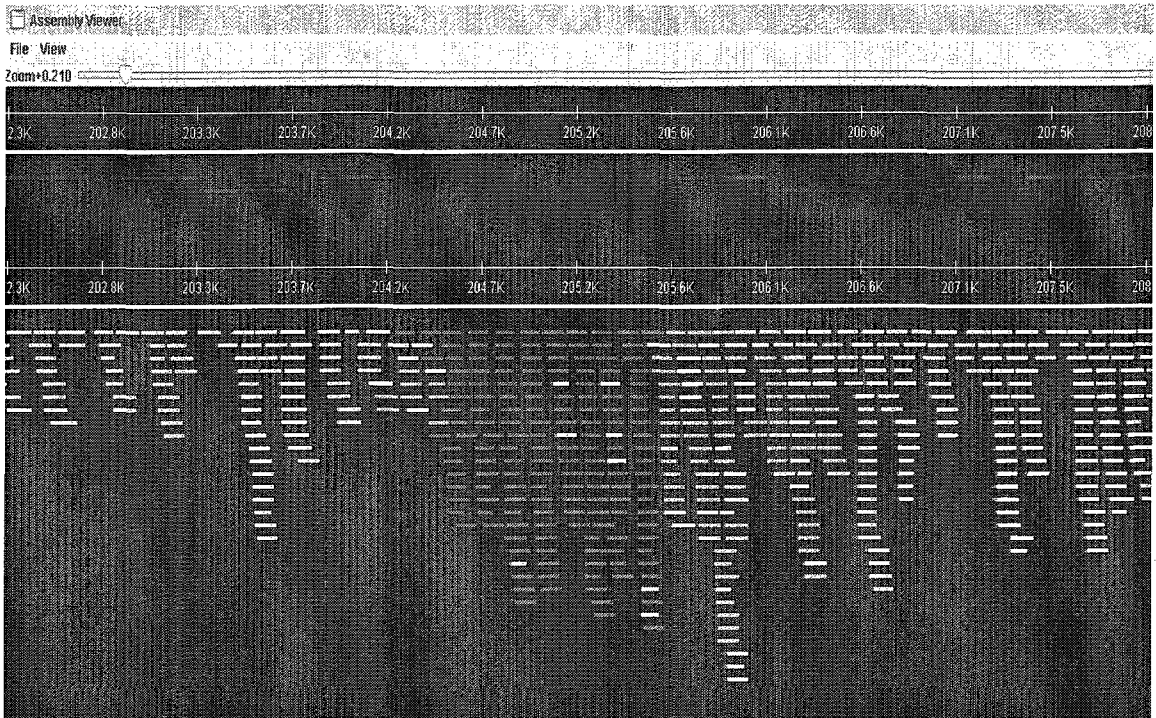


Figure 4.5-5 A MAVEN display of results from a comparative assembly.

Figure 4.5-5 very well conveys the benefits gained by MAVEN for viewing the comparative assembly. White lines represent the reference genome. Blue lines are the assembled contigs. The other lines are input sequences aligned to the reference and color-coded to represent the number of valid alignment locations for that sequence.

The contigs formed and the aligned reads can be viewed at the same time. It can be clearly noted that the assembly failed to form a contig between 204.2K and 205.6K because the reads in that region were aligning at 6 or more different places within the reference region. This indicates that the red region is a repetitive region in the reference sequence and hence the comparative assembly failed to generate contigs in this region.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The initial goals set for the research were to develop comparative genomics tools that could handle 454 sequence data. Accordingly, we identified three software tools to be developed:

- Mutation Detector should be able to identify and confirm mutations across the different strains of organisms.
- MAVEN should be able to display multiple assemblies simultaneously. It should also be able to display the results of *de novo* assembly and alignments.
- Iterative Multi-Genome Comparative Cross Assembly Processor might provide a framework for better assembly of multiple related genomes in the absence of a common reference.

The status of each of the three tools is summarized in the sections that follow.

5.1 Mutation Detector

The Mutation Detector is completely implemented. It can detect mutations across strains when compared to the reference sequence. It took approximately 55 seconds to produce the output files for genomes that were approximately 200,000 base pairs in length. For genomes that were approximately 1,400,000 base pairs long, it took

approximately 4 minutes to produce the output files. After the output of MD is obtained, it takes around 2-3 minutes to produce the error calculation tables.

The mutation detection process is not limited to reads that are sequenced using the 454 sequencing technology. The traditional sequence reads can take advantage of this method too. As long as there is a reference sequence available for comparative assembly, this method can prove to be very effective in finding mutations and putations occurring along any MA line.

5.2 MAVEN

MAVEN is able to display multiple assemblies simultaneously for comparative as well as *de novo* assembly. It can also display sequence alignments. From the results of comparative assembly combined with the sequence alignments for the same assembly, MAVEN is able to determine the highly repetitive regions within the reference genome. It can also determine the reads that aligned to the reference but are not used in the contig formation process.

5.3 Iterative Multi-Genome Comparative Cross Assembly

Processor

To carry out an iterative comparative assembly of sequence reads, it is imperative to find a *de novo* assembler which is optimized for 454 data for the first step. We tried to use the TIGR *de novo* assembler [9, 12] to assemble the sequence data of the 4 yeast strains. The TIGR assembler is developed with the goal of being used with the traditional shotgun sequence reads. To be useful for 454 data, a complete analysis and understanding of the assembly algorithm is essential. We evaluated the use of this assembler with our data by setting up different assembly parameters. It was not very fruitful since the contigs

that were formed after the assembly were not long enough to be useful for further assembly.

After a lot of experimentation with assemblers used for assembling different strains of yeast sequenced using the 454 technology, it has become evident that the AMOScmp comparative assembler is well suited for assembling the sequence reads, provided a closely related reference species is available. We tried multi-genome iterative cross assembly idea using AMOScmp, but the preliminary results were not very promising.

The *de novo* assembly of the individual strains was only able to generate contigs that averaged about twice the size of the input sequences. Also, we were not able to significantly increase the contig sizes using comparative assembly. We believe that the iterative cross assembly idea may still have merit, but may need better *de novo* assemblies as starting points.

LIST OF REFERENCES

1. Delcher, A.L., S. Kasif, R.D. Fleischmann, J.Peterson, O. White and S.L. Salzberg Alignment of whole genomes, *Nucleic Acids Research*, 1999, 27(11):2369-2376
2. Drake, J., B. Charlesworth, D. Charlesworth, and J. F. Crow. 1998. Rates of spontaneous mutation. *Genetics* 148: 1667-1665
3. Friedberg, E. C., G. C. Walker, and W. Siede. 1995. DNA Repair and Mutagenesis. ASM Press, Washington, D. C.
4. Kellis, M., Patterson, N., Endrizzi, M., Birren, B., and Lander, E.S., 2003. Sequencing and Comparison of Yeast Species to Identify Genes and Regulatory Elements. *Nature* 423:241-254.
5. Li, W.-H. 1997. Molecular Evolution. Sinauer Assocs., Inc. Sunderland, MA.
6. Maki, H. 2002. Origins of spontaneous mutations: specificity and directionality of base-substitution, frameshift, and sequence-substitution mutageneses. *Ann. Rev. Genet.* 36: 279-303.
7. Margulies, M. et al. Genome sequencing in microfabricated high-density picolitre reactors, *Nature* 437:376-80, 2005.
8. Pop, M. Shotgun sequence assembly. *Advances in Computers* vol. 60, M. Zelikowitz ed. June 2004.
9. Pop, M., D. Kosack. Using the TIGR Assembler in shotgun-sequencing projects. in *Bacterial Artificial Chromosomes* vol. 1, S. Zhao and M. Stodolsky eds. Humana Press, pp. 79-294, March 2004.
10. Pop, M., A. Phillippy, A.L. Delcher, S.L. Salzberg. Comparative Genome Assembly. *Briefings in Bioinformatics*, 2004. 5(3):237-48. (AMOS)
11. Pop, M., S. L. Salzberg, M. Shumway. Genome Sequence Assembly: Algorithms and Issues. *IEEE Computer* 35(7) 2002, pp. 47-54.

12. Sutton, G.G, O. White, M. Adams, and A.R Kerlavage. TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Science and Technology*, 1:9-19, 1995.

APPENDIX A

MUTATION DETECTOR (MD) DOCUMENTATION

Installation:

- Install Java JDK on the computer
- Perl, csh and tcsh need to be installed on the computer to run perl scripts.

MD steps and scripts:

The MD tool is made up of a Java program and a bunch of perl scripts that aid us in generating the analysis tables mentioned in section 3.3. We have programs/scripts written to carry out all the different steps mentioned in sections 3.3.1- 3.3.10. Summary of the steps follows:

- Creation of a delta file for each strain – this is done by the comparative assembly process (*AMOScmp*).
- Alignment Summary – this is where we create a summary of alignments for each comparative assembly. This is done by the java program *SeqAlignGenerator* and the perl scripts *align-summary*, *getrefPosnBase* and *combineRefDelta*.
- Processing Alignment Summary – this is where we remove the insertions, the homopolymer region of 4 or more and 5 bases around the homopolymer region. This is made possible by the perl scripts *removeInsertions*, *delHomopolymers*, *getHPposns*, *addStrainName* and the java program *DelByName*.
- Combining MD output files, applying coverage restrictions and creation of analysis tables – all these steps are done by the perl script *sh_error*

- Combined analysis – when a combined analysis is done for all the 4 strains together as mentioned in section 3.5, the perl script *strainCombineAnalysis* aids in combining the data for the 4 strains. Then the perl script *sh_error* is used to generate the analysis tables for the combined analysis.

What follows is a detailed description of how these scripts are executed.

1. **java SeqAlignGenerator <delta file> <sequence file> > outDeltaFinal**

- SeqAlignGenerator* – java program that generates a read alignment from the delta file. A row in the output file that it generates consists of the reference base position, the read id that aligned at that position and the read base.
- <delta file >** - delta file generated for the MA line by the comparative assembly is an argument to the script.
- <sequence file>** - seq file generated for the MA line by the comparative assembly is passed as an argument to the script.
- outDeltaFinal** – the output of the *SeqAlignGenerator* is redirected to a file which we call outDeltaFinal. The rows are sorted according to the read id (integer value).

2. **cat outDeltaFinal | sort -g > sortedResult**

- cat* – This is a unix command to print a file.
- outDeltaFinal** – the output file from step# 1.
- sortedResult** – we sort the data in outDeltaFinal file based on the reference position and store this sorted data in sortedResult file. Note that the ‘reference base position’ values will not be unique in the column – they will appear as many times as there are reads aligning at that particular position.

3. *align-summary* sortedResult > sortedResultSummary

- a. *align-summary* – a perl script that generates the alignment summary for the strain. The output that it generates consists of 2 columns- ‘reference base position’ and ‘aligned read bases’. The ‘aligned read bases’ column has the bases of all the reads that align to that particular position. It basically takes the output file from step#2 and makes the values in the ‘reference base position’ column unique by summarizing all the bases of the reads that align at that position in the same row.
- b. **sortedResult** – output file from step#2 is passed in as an argument to this script.
- c. **sortedSummaryResult** – output generated by the *align-summary* script is redirected to this file.

4. *getRefPosnBase* refFile > refBases

- a. *getRefPosnBase* – perl script that reads in a reference sequence and generates an output that has 2 columns: ‘reference base position’ and ‘reference base’.
- b. **refFile** – fasta file consisting of the reference sequence is an argument to the script..
- c. **refBases** – the output from the perl script is redirected to this file.

5. *combineRefDelta* refBases sortedSummaryResult > deltaOut

- a. *combineRefDelta* – perl script that combines the data from the file refBases (from step#4) and sortedSummaryResult (from step#3). The output of this script has 3 columns – ‘reference base position’ (unique positions),

'reference base' (from refBases file) and 'aligned read bases' (from sortedSummaryResult file).

- b. **refBases** – the output file from step#4 is passed as an argument to this script.
- c. **sortedSummaryResult** – the output file from step#3 is passed as an argument to this script.
- d. **deltaOut** – the output of the perl script is redirected to this file.

6. *removeInsertions* deltaOut > strainDelta

- a. *removeInsertions* – perl script that removes the insertion positions from the deltaOut file generated in step#5
- b. **deltaOut** – this file is passed as an argument to the script
- c. **strainDelta** – the output of the script is redirected to this file.

7. *delHomoPolymers* strainDelta > HPs

- a. *delHomoPolymers* – this perl script aids in finding the homopolymer regions in the reference sequence.. The output of this script gives a range of consecutive base positions that have the same base. For example, if the 'strainDelta' file has data as shown in Table 3.3-3, this script will generate a single column with the following values (one value per row) – 50-50, 51-52, 53-53, 54-54, 55-55, 56-59, 60-60, 61-62, 63-63, 64-64, 65-65,66-66. Using this information, we can see that there is a homopolymer of length 2 (base positions 51-52) and a homopolymer region of 4 bases (base positions 56-59).

- b. **strainDelta** – the output from step#6 is passed in as an argument to this script.
 - c. **HPs** – the output of this script is redirected to this file.
8. ***getHPposns* HPs <homopolymer length> > homopolymerBasePositions**
- a. ***getHPposns*** – this perl script gets the homopolymer region base positions for the homopolymer regions in the file HPs (generated in step#7).
 - b. **HPs** – output file from step#7
 - c. **<homopolymer length>** - we want to detect the homopolymers greater than the number provided by this argument. In our analysis, since we want to delete all the homopolymers of length greater than 3, we pass in the integer value '3' as the second argument to this script.
 - d. **homopolymerBasePositions** – the output of this perl script is redirected to this file.
9. ***java DelByName* homopolymerBasePositions strainDelta > strainDeltaCopy**
- a. ***DelByName*** – this java program helps in deleting all the homopolymer regions of length 4 or more and also 5 base positions surrounding such homopolymer regions from the strainDelta file generated in step#6.
 - b. **homopolymerBasePositions** – this file generated in step#8 tells which base positions need to be deleted so that we eliminate the homopolymer region of 4 or more.
 - c. **strainDelta** – the homopolymer region of 4 or more needs to be deleted from this file.

- d. **strainDeltaCopy** – the output from this java program is redirected to this file.

10. ***addStrainName* strainDeltaCopy <strain name> > finalStrainData**

- a. ***addStrainName*** – this perl script appends the strain name as a column to the strainDeltaCopy file generated in step#9.
- b. **strainDeltaCopy** – output file generated ins stel#9
- c. **<strain name>** - name of the strain to be appended for example A1, A4 etc.
- d. **finalStrainData** – the output of the perl script is redirected to this file. This file has all the strain specific data that we need – the alignment summary without the insertions and the homopolymer region. One such file is created per strain. This file is then used to generate the analysis tables.

11. ***sh_error* <finalStrainData 1> <finalStrainData 2> <min coverage> <max coverage> > analysisTable**

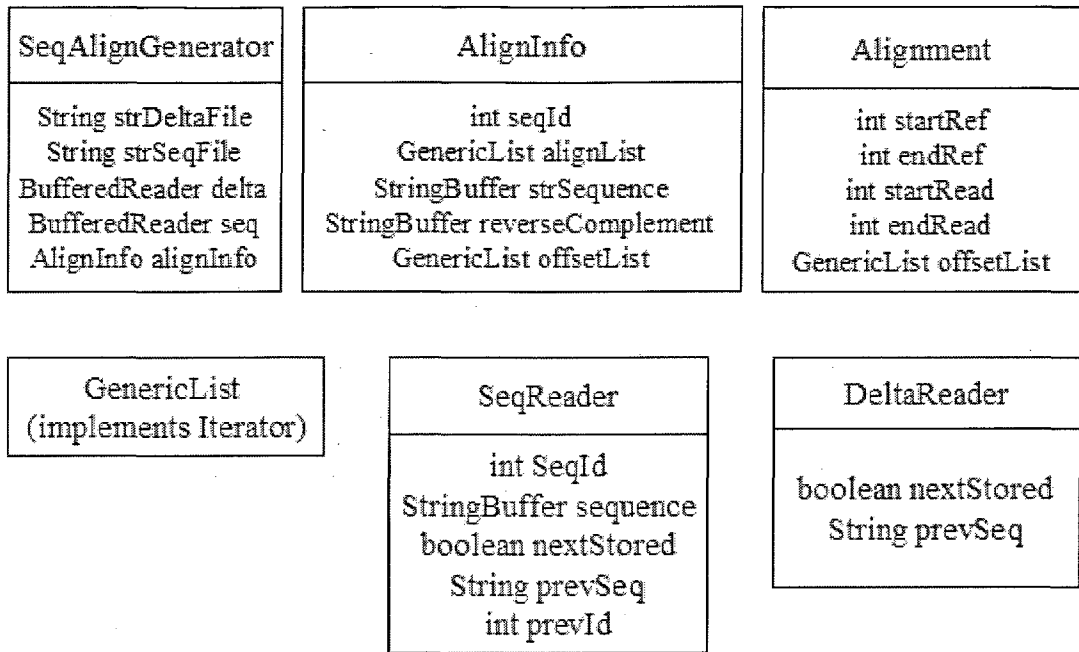
- a. ***sh_error*** – this perl script is responsible for generating the analysis tables for the 2 strains whose ‘finalStrainData’ files we pass in as arguments.
- b. **<finalStrainData 1>** - the ‘finalStrainData’ file generated for the first strain (one of A1, A4, C5 or C8). This ‘finalStrainData’ file is generated for that particular strain in step#10.
- c. **<finalStrainData 2>** - the ‘finalStrainData’ file generated for the second strain (one of A1, A4, C5 or C8).
- d. **<min coverage>** - for a 2-strain or all-strain analysis the minimum coverage for our study is ‘3’.

- e. **<max coverage>** - for a 2-strain analysis, the max coverage should be '10'.
For all-strain analysis, the max coverage should be '30'.
- f. **analysisTable** – the output for this script is captured in this file.

12. *strainCombineAnalysis* ref **<finalStrainData 1>** **<finalStrainData 2>**
<finalStrainData 3> **>** **combinedStrainData**

- a. *strainCombineAnalysis* – this perl script is used when we want to analyze one strain against the combination of the other three strains. The 'finalStrainData' of the other three strains needs to be combined before we execute the *sh_error* script on those.
- b. **ref** – the reference sequence is passed as an argument to this script.
- c. **<finalStrainData 1>**, **<finalStrainData 2>** and **<finalStrainData 3>** - the 'finalStrainData' files for the three strains that needs to be combined.
- d. **combinedStrainData** – the output of this script is captured in this file.
- e. *sh_error* **<finalStrainData 4>** **combinedStrainData 3 30 >** **analysisTable**
– this is how we execute the *sh_error* perl script when we want to do an all-strain analysis.

Class Diagram



Class Diagram for SeqAlignGenerator

Class Descriptions:

1. **SeqAlignGenerator**: This is the main class used to generate the alignment information from the '.delta' and '.seq' files. It accepts these 2 files as input.
2. **AlignInfo** – This class is used to store the alignment information for a single read present in the delta file. It consists of the sequence id, the actual string sequence and the reverse complement. It also stores a list of alignments for the particular read.
3. **Alignment** – This class is used to store the information of a single alignment for a read.
4. **DeltaReader** – This class is used to read the delta file and create the alignment object.

5. **SeqReader** – This class is used to read the seq file.
6. **GenericList** – This class is used to construct a list of objects (Object type).

APPENDIX B

MAVEN DOCUMENTATION

Installation:

- Install Java JDK on the computer
- Perl, csh and tcsh need to be installed on the computer to run perl scripts.

MAVEN Usage

- To launch the MAVEN tool
 - Open a command prompt
 - Go to the directory where the MAVEN software is deployed (the directory that has all the class files generated by compiling the tool).
 - From this directory, run the command “java *AsmRenderer*”.

In the visual tool, you can view the results of three assembly types which are discussed below.

GUI Navigation:

The MAVEN tool, can display three different types of assembly results:

- ***De-novo Assembly*** – A *de-novo* assembler produces contig files as a result of the assembly. These contig files give the information about the number of sequence reads that make up the contig along with their alignments. To display the *de-novo*

assembly results, the data from the contig file is converted to a format suitable for MAVEN. The converted format is placed in a file with an extension “.denovo”. The data format is converted using a perl script *getContigReadInfo*. The contig file has information in the following format shown below.

```
##1 chr1G1 18 462 bases, 00000000 checksum.  
#128698_3899_0794(0) [RC] 118 bases, 00000000 checksum. {116 1} <1  
118>
```

This is converted to a format as shown below by the *getContigReadInfo* perl script.

```
Contig 1 462  
Read 128698_3899_0794 1 118
```

The *getContigReadInfo* perl script is executed as follows.

```
getContigReadInfo <contig file> > <output file name>.denovo
```

<contig file> - name of the contig file generated by the assembly. Note that the file name should be specified with the “.contig” extension.

<output file name> - output of the perl script should be redirected to a file with “.denovo” extension. The output file has the assembly information that can be parsed by MAVEN.

To see the *de-novo* assembly results inside MAVEN:

- Open the ‘File’ menu on the left hand side of the tool.
- Select your denovo output file (<output file name>.denovo).

The data from the file is parsed by MAVEN and the assembly result is displayed in the MAVEN GUI.

- **Comparative Assembly** – A comparative assembler (*AMOScmp*) generates a “layout” file. This layout file has all the information about the contigs and the

different reads that make up the contig. The *getLayoutInfo* perl script is used to convert the data in the layout file so that it can be parsed properly by MAVEN. The *getLayoutInfo* script is executed as follows.

***getLayoutInfo* <layout file> <reference length> > <output file>.comp**

<layout file> - layout file generated by the comparative assembler. The file name should be specified with its extension “.layout”.

<reference length> - length of the reference sequence must be provided

<output file> - output of the perl script should be redirected to a file with “.comp” extension.

To see the comparative assembly results inside MAVEN:

- Open the ‘File’ menu on the left hand side of the tool.
- Select your comp output file (<output file name>.denovo).

The data from the file is parsed by MAVEN and the assembly result is displayed in the MAVEN GUI.

- **Comparative Assembly Alignments** – A comparative assembler (*AMOScmp*) generates a “delta” file during assembly. This file has the information about all the reads and their alignments with respect to the reference genome. It is not necessary that all of these reads are part of a contig. The alignment information is very important to determine the individual read alignments, especially for the reads that are discarded from the contig creation step. The *getCompleteAlignment* perl script is used to convert the data in the “delta” file to aid parsing. The *getCompleteAlignment* script is executed as follows.

***getCompleteAlignment* <delta file> <output file>.align**

<delta file> - delta file generated by the comparative assembler. The file name should be specified with extension.

<output file> - output file name should be specified as an argument to the script. It should have an extension “.align”.

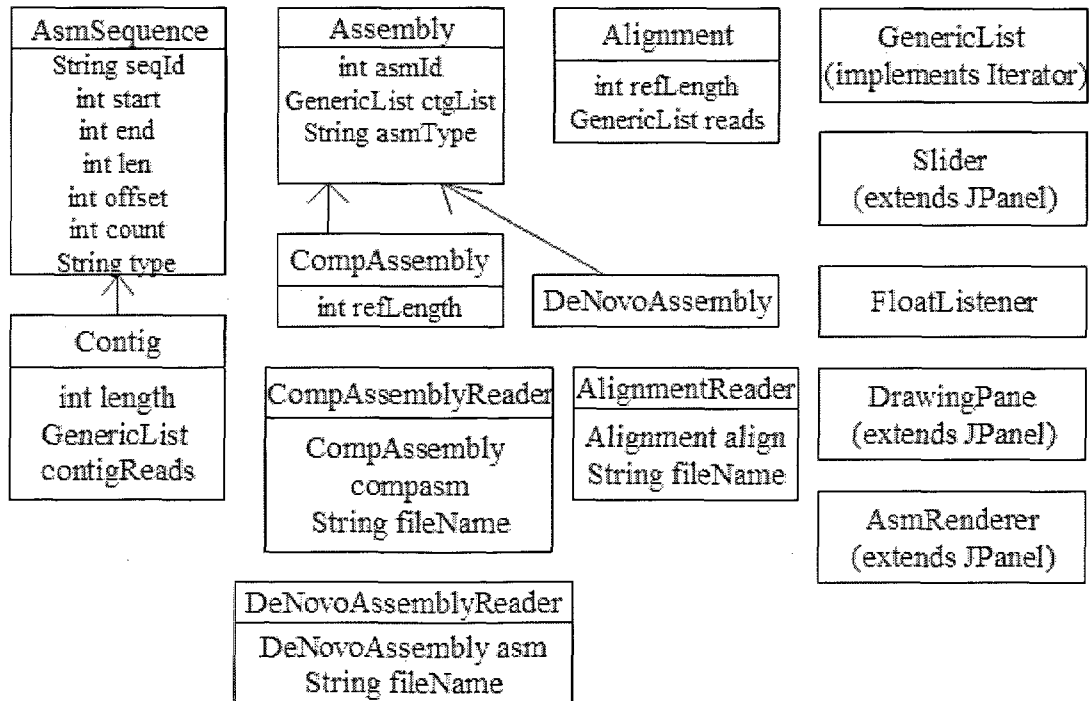
To see the comparative assembly alignment results inside MAVEN:

- Open the ‘File’ menu on the left hand side of the tool.
- Select your “.align” output file (<output file name>.align).

The data from the file is parsed by MAVEN and the alignment result is displayed in the MAVEN GUI.

Implementation Details: Class Diagram

The figure below shows the class diagram of the classes used in MAVEN



Class Diagram for MAVEN

Class Descriptions:

1. **AsmSequence:** This class represents a generic sequence. It can be either a contig or a sequence read. It consists of the following member variables:
 - a. **sequenceId** - unique identifier for the sequence
 - b. **start and end** – these represent the start and end offset when this sequence aligns with its reference. If the sequence represents a “sequence read”, then start and end offsets tell how this sequence read aligns with a contig. If the sequence represents a “contig”, then start and end offsets tell how this contig aligns with a reference sequence for a comparative assembly.
 - c. **offset** – this is used for display purposes.
 - d. **count** – keeps a count of how many times this sequence aligns at different positions in the reference sequence.
 - e. **type** – this value indicates whether the sequence is “sequence read” or “contig”
2. **Contig** – This class extends from AsmSequence and represents a contig in an assembly. A contig has a length and a list of sequence reads associated with it.
3. **Assembly** – This class represents an assembly object. It has the following member variables:
 - a. **asmId** – unique identifier for the assembly
 - b. **ctgList** – list of contigs that make up this assembly
 - c. **asmType** – a string representing what type of assembly this is: comparative or *de-novo*.

4. **CompAssembly** – This class represents a comparative assembly object. It extends from the `Assembly` class. It defines an additional member variable called “`refLength`” that represents the length of the reference sequence.
5. **DeNovoAssembly** – This class represents a *de-novo* assembly object and also extends from the `Assembly` class.
6. **Alignment** – This class represents an alignment object used to display the alignment results for a comparative assembly. It is made of a list of aligning reads and the length of the reference sequence. Note that it does not have any contigs since the alignment is just for the sequence reads with respect to the reference sequence.
7. **CompAssemblyReader** – This class helps in rendering the results of a comparative assembly. It reads in the data file that has the information about the comparative assembly and renders the information in the GUI.
8. **DeNovoAsmReader** – This class helps in rendering the results of a *de-novo* assembly. It reads in the data file that has the information about the *de-novo* assembly and renders the information in the GUI.
9. **AlignmentReader** – This class helps in rendering the results of read alignments for the comparative assembly. This reads in the data file that has the information about the alignments and renders the information in the GUI.
10. **GenericList** – This class implements the `Iterator` and is used to construct a list of objects (`Object`) and iterate over them. For MAVEN, we use this class to represent a list of contigs, list of sequence reads and list of assemblies to be displayed in the GUI.

11. **Slider** – This class supports a JSlider with a label and an arbitrary range of floating point values. This class is used to implement the “zoom” functionality in MAVEN. It is implemented by RDB.
12. **FloatListener** – This class allows us to connect to a Slider. This class is used to implement the “zoom” functionality. It is implemented by RDB.
13. **DrawingPane** – This class extends from JPanel and provides the panel inside which all the assembly objects are displayed.
14. **AsmRenderer** – This class is the driving class for the assembly viewer. It is the holder of the drawing pane, the slider and all the different parts that make up the viewer.