Appendix C

Sequence Analysis

Python and shell scripts were written to execute a sequence analysis pipeline on the Harvard University Odyssey research computing cluster. Scripts were also written to find genomic copy number variations and plot the results. The details of what these scripts do and how to run them are described below. The files described in this appendix are available at https://github.com/nwespe/sequence_analysis/.

C.1 Analysis pipeline

The pipeline is executed by a series of commands in the Bash script "segtools" written by John Koschwanez, which was modified to include additional processing steps and to accommodate parallel sample analysis. The execution of these commands was divided into three parts to enable parallel processing. The first part takes raw FASTQ files as received from the Bauer Sequencing Core Facility, i.e., already demultiplexed, and aligns

the sequence data to a reference genome, generating pileup files. The second part identifies mutations by comparing the pileup files to a reference genome. The third part locates the mutations relative to known features on the reference genome using a script written by John Koschwanez, "mutantanalysis.py." The software used, the input files, and the output files are described below for each part. The method of running this analysis pipeline using the Python scripts I wrote is described in Section C.2.

Potential future updates to the pipeline include updating the versions of the software programs used and adding a step to remove PCR duplicates from the analysis. There is room for improvement in the handling of sequences that map to more than one location in the reference genome; see Section 3.4 for the trouble that can arise from this. Detection of structural variation is absent from the pipeline described here; Miguel Coelho has used the program SoftSV to analyze chromosome rearrangements and this step could be added to the existing pipeline. Programs for running SoftSV and Breseq in parallel on the cluster were written and used by others but not extensively tested and are not described here.

I. Generate alignment

INPUT. The FASTQ data file contains four lines of information for each read; one example is rearranged into numbered lines below. Each read has a unique identifier including information on the machine used to generate the data and the adapter sequence used for multiplexing (lines 1-2). Following that is the sequencing read (lines 3-5) and the associated quality score for each base read (lines 7-9), separated by a line containing a '+' symbol. More information on the FASTQ format can be found in online resources.

```
1  @ILLUMINA-D00365:412:HCMTVADXX:2:1116:17461:37814 1:N:0:TAG
2  GCATGGTAAGGAG
3  CCAAAGACAATAGAAGCTTCAAAAGAGTCACTTAAACCGACAGCCTGGAAAACAATGGT
4  ACCGTAGTAGAAGAAATAGTTATCACCAGTTAATTGTTGTAGAGATTGAATCATGATAC
5  CCATCATAGTACGTTGAAACATGGCTGGTTTA
6  +
7  CCCFFFFFHHGHHJJJJJJJJJJJJIJJHHIJJJJJJJJJGIIIIIIJJJJJJJJJJJJJ
8  HIJHHHGEFFFFFEEEEEDDEEEDEDDDDCDDEEEDCDDDDDACDDDDDDDCDDDDDDC
9  CDDDDDDDDEEEDDDBDDDDDDCCDDDDDDBD
```

Step 1: adapter trimming. The cutadapt program searches each read for sequences matching the adapters used to generate the library and removes these bases, and any following bases, from the read. The adapter sequences are hard-coded into the segtools program and can be changed there if necessary. This step generates a new FASTQ file for each file analyzed and a report for each file or pair of files, in the case of paired-end reads; a part of the report is excerpted below.

```
Total read pairs processed:  1,315,728
  Read 1 with adapter:  728,181 (55.3%)
  Read 2 with adapter:  725,430 (55.1%)
Pairs written (passing filters):  1,315,728 (100.0%)

Total basepairs processed:  394,718,400 bp
  Read 1:  197,359,200 bp
  Read 2:  197,359,200 bp
Total written (filtered):  329,882,059 bp (83.6%)
  Read 1:  164,886,064 bp
  Read 2:  164,995,995 bp
```

Step 2: quality control. The FastQC program generates an HTML report providing an overview of several quality checks of the read data, such as per-base sequence quality, sequence length distribution and overrepresented sequences. A version of this report is also provided by the Bauer Core Facility with the FASTQ files, so it is possible to compare this information before and after adapter trimming.

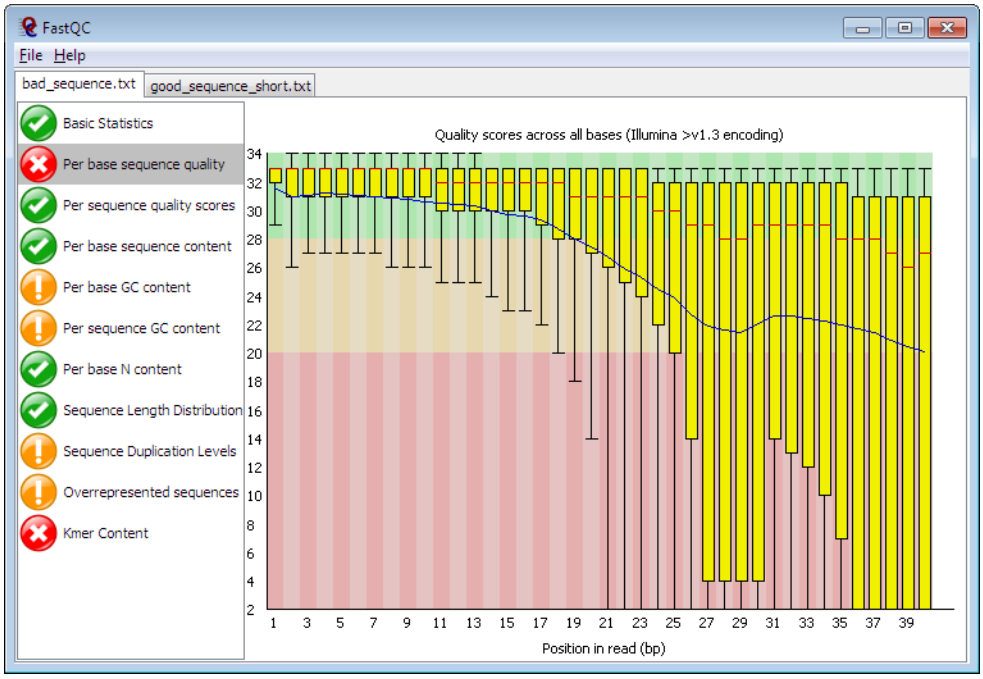


Image from https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Step 3: alignment to reference. The program used for aligning the sequence reads to the reference genome is BWA-MEM. The output is piped directly into several functions in SAMtools for sorting and indexing, which produces a sorted BAM (Binary sequence Alignment/Map) file. The sorted BAM file is processed by the 'IndelRealigner' function from the Genome Analysis ToolKit to improve alignment around insertions and deletions; this produces another BAM file. In SAM/BAM format, each read occupies one line; one example is rearranged into numbered lines below. The reads are ordered according to the reference coordinates of their primary alignment.

```
1   ILLUMINA-D00365:412:HCMTVADXX:2:2101:4568:67853   163
2     ref|NC_001133|   531   60   150M   =   565   184
3   CATCATTATGCACGGCACTTGCCTCAGCGGTCTATACCCTGTGCCATTTACCCATAAC
4   GCCCATCATTATCCACATTTTGATATCTATATCTCATTCGGCGGTCCCAAATATTGTA
5   TAACTGCCCTTAATACATACGTTATACCACTTTT
6   CCCFFFFFGHHHHJJJJIJJJJJIJJGIIIHHIIIJJIJIJHIJJFJJJJJIJJJJJJ
7   JJHHFFFFEEEEDEEDDDDFEDBDDFDFEFFEEFDDEEEEDBBDDBDBDCDDDEEECC
8   DDEDCDDDDDDCDEDDCDDCDCCDBDCCDDDDDD
9   XA:Z:ref|NC_001135|,+828,150M,5; MC:Z:150M MD:Z:150
10  RG:Z:028A NM:i:0 MQ:i:60 AS:i:150 XS:i:125
```

The read information begins (lines 1-2 above) with the unique identifier from the FASTQ file followed by a bitwise FLAG value (`163`), the reference coordinate (`ref|NC_001133| 531` indicates Chromosome 1, base 531) and the mapping quality (`60`). Next is a CIGAR sequence (`150M`) that describes the alignment; in this case, the entire read of 150 bases was aligned (M denotes match/mismatch). The next part, `= 565 184`, indicates the coordinates of either the next read or the mate read (same chromosome, base 565) and the observed template length (i.e., insert length) determined from the mate's relative alignment (184). This information is followed by the sequence (lines 3-5), the base quality scores (lines 6-8), and a series of optional fields (lines 9-10). For interpretation of the bitwise FLAG value and the optional fields, see other SAM documentation.

In paired-end sequencing data, each read has a mate. The first part of the read information for mate of the example above is shown below.

```
1   ILLUMINA-D00365:412:HCMTVADXX:2:2101:4568:67853   83
2     ref|NC_001133|   565   60   150M   =   531   -184
```

The template length of the mate is given the opposite sign (`-184`). Because each read is 150 bases, a template length of less than 300 indicates that some bases of the template were read twice. This is accounted for by the next processing step.

Step 4: soft-clip overlapping bases. For paired-end data, the realigned BAM file is passed to the 'clipOverlap' function from BamUtil; this function identifies bases that were sequenced in both reads of a pair and 'soft-clips' one read to exclude the overlapping bases from downstream analysis. A new BAM file is created in which the CIGAR sequence for the read has changed to indicate which bases were clipped. In the example below, the 150 aligned bases are now 34 aligned bases followed by 116 soft-clipped bases. The BAM file still contains the sequence and quality data for these bases, but they will not be used to generate the pileup file in the next step.

Before clipping:

```
1   ILLUMINA-D00365:412:HCMTVADXX:2:2101:4568:67853   163
2     ref|NC_001133|   531   60   150M   =   565   184
```

After clipping:

```
1   ILLUMINA-D00365:412:HCMTVADXX:2:2101:4568:67853   163
2     ref|NC_001133|   531   60   34M116S   =   565   184
```

Step 5: generate pileup file. The final BAM file is processed by the SAMtools 'mpileup' function to create a pileup file. This file compiles the bases mapped to each location in the reference. Four lines are excerpted from a pileup file below.

```
ref|NC_001133| 534 C 50
  ,,.,.................,.,.,............,$,,,.,...^K,
  FJCIDDCEEFFFFHJJJJJJDIJDJJBJGGJJIJIGJIJBJHCGHFFCCC
ref|NC_001133| 535 A 49
  ,,.,.................,.,.,...........,,,.....,
  FJCGCDEECFFEEHIJJJGIFHJDJIBJIIIJJJGEJJGIHCHEFFCCC
ref|NC_001133| 536 T 49
  ,,.,.................,.,.,...........,,,.....,
  FJCIDDDDEFFEEHIIHJIGDIIDJIBJIGIJIJIIJJIGDAHEDFFFC
ref|NC_001133| 537 T 49
  ,cCc.....CC...CC.C.C,..,.cc..CCCCC.....,cc.c....c
  FJDFEEEEEFFEEFIJHJJHDJJDIJFJJJIJJJJIJJIHGCHIFFFFC
```

Each line begins with the reference sequence coordinate (in this case, Chromosome 1, bases 534 through 537) followed by the reference base and the number of reads mapped to that location. The string of symbols indicates whether the mapped read is a match to the forward (.) or reverse (,) strand, or a mismatch to the forward (ACTGN) or reverse (actgn) strand. The symbol ^ marks the beginning of a read, the character following it indicates the read's mapping quality, and the symbol $ marks the end of a read. The string of letters are the corresponding base qualities.

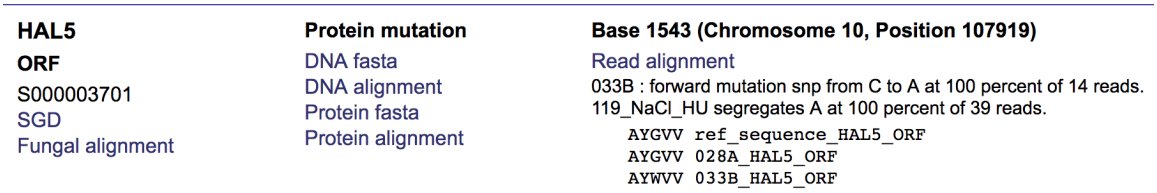
Find mutations in clones. The analysis pipeline finds mutations in clones using the VarScan 'somatic' function. VarScan was written to find sequence variants (SNPs and indels) between normal and tumor samples, and it classifies variants as either germline (both samples differ from the reference genome) or somatic (only the tumor differs). When identifying mutations in evolved strains, the evolved clone is considered the tumor sample and the ancestor is the normal sample. The first 15 columns of three lines are excerpted from a VarScan .snp file below. `normal/tumor_reads1` and `reads2` are the reads with the reference (`ref`) and variant (`var`) base, respectively; `normal/tumor_gt` is the consensus genotype.

```
chrom           position  ref  var  normal_reads1  normal_reads2
ref|NC_001133|  537       T    C    29             20
ref|NC_001133|  610       G    A    18             13
ref|NC_001142|  107919    C    A    14             0

normal_var_freq  normal_gt  tumor_reads1  tumor_reads2  tumor_var_freq
40.82%           Y           36            26            41.94%
41.94%           R           26            32            55.17%
0%               C           0             14            100%

tumor_gt  somatic_status  variant_p_value          somatic_p_value
Y         Germline        3.7752715862120977E-17   0.5304477608132048
R         Germline        1.4812831017457745E-17   0.16669057568084028
A         Somatic         1.0                      2.4927336813189342E-8
```

Find mutations in populations. The analysis pipeline finds mutations in populations using the VarScan 'pileup2snp' and 'pileup2indel' functions, which compare a single pileup file to a reference genome. This function outputs similar information as the VarScan 'somatic' function but cannot distinguish pre-existing mutations from evolved mutations (i.e., germline versus somatic). These files are used in conjunction with the ancestor-versus-evolved clone VarScan files generated by the 'somatic' function to identify evolved mutations present at high frequency in either a back-crossed pool or an evolving population. Thirteen columns of three lines are excerpted from a VarScan .snp file below.

```
Chrom           Position  Ref  Cons  Reads1  Reads2  VarFreq  Strands1
ref|NC_001133|  537       T    Y     77      42      35.29%   2
ref|NC_001133|  610       G    R     40      28      41.18%   2
ref|NC_001142|  107919    C    A     0       39      100%     0

Strands2  Qual1  Qual2  Pvalue                   ...  VarAllele
2         38     38     2.750496826499323E-15    ...  C
2         36     36     1.1003828151779401E-10   ...  A
2         0      37     3.6741722220670473E-23   ...  A
```

Classify mutations. The final step of the automated pipeline uses the "mutantanalysis.py" script written by John Koschwanez. This script matches the sequence variants identified by VarScan to annotated features on the reference genome. It determines whether the mutation affects coding or non-coding DNA and, if in a coding region, the effect of the mutation on the amino acid sequence. It outputs this information in an HTML file with links to the Saccharomyces Genome Database and in two text files that list the information by gene and by mutation. It also compiles the reads from the BAM file that overlap the mutated region and enables viewing them in the HTML file.



Tabbed output by gene:

gene_name	sgdid	chr_num	position	sample_name	snp_indel
HAL5	S000003701	10	107918	033B	snp
WHI2	S000005569	15	412304	033B	snp
IRC23	S000005570	15	412304	033B	snp

ref	read	fraction	tot_reads	mutation_type
C	A	100	14	nonsynonymous
A	G	100	10	nonsynonymous
A	G	100	10	promoter

Tabbed output by mutation:

chr_num	position	snp_indel	seg_percent	genes	sample_name
10	107918	snp	100	HAL5	033B
15	412304	snp	77	WHI2,IRC23	033B

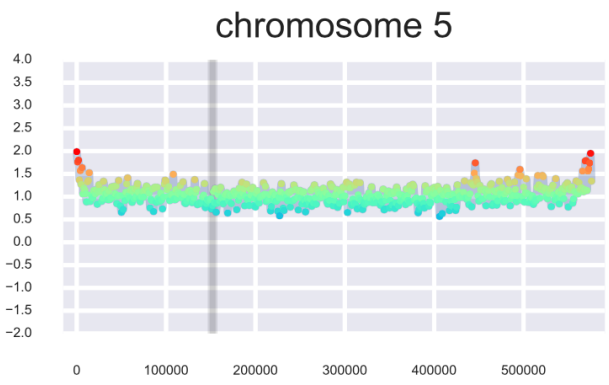
Determine likelihood of causality. This is where the human analysis takes over. The user must evaluate the likelihood of the identified mutations to cause the phenotype of interest. This analysis could take into consideration the effect of the mutation on the amino acid sequence, the previously-described function of the gene product, and any phenotypes associated with mutations in the locus. In the case of bulk segregant analysis, it is important to consider whether there is linkage to other loci under selection, e.g., a locus used to generate the pool of spores.

Measure copy number variation. The script "copynumber.py" finds differences in copy number between two pileup files using the VarScan 'copynumber' function. Three lines are excerpted from a resulting .copynumber file below.

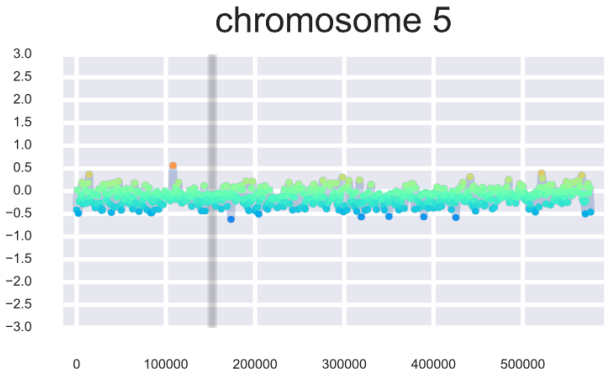
```
chrom           chr_start    chr_stop    num_positions
ref|NC_001133|  9372         9471        100
ref|NC_001133|  9472         9571        100
ref|NC_001133|  9572         9671        100

normal_depth    tumor_depth  log2_ratio  gc_content
28.7            22.1         -0.376      39.0
27.2            21.3         -0.352      37.0
20.4            24.0         0.232       32.0
```

Visualize copy number variation. The copy number data can be visualized using the script "plot_copynumber.py" for one sample or the script "batch_plot_copynumber.py" for multiple samples. Each plot displays read depth normalized to the entire genome versus chromosome position. The resulting plots for all chromosomes of a sample are saved in a single PDF. The script generates one file for each sample plus one file for each comparison (e.g., ancestor versus evolved) listed in the input text file; for the latter, the read depth of the evolved sample is divided by that of the ancestor.

Read depth of a single sample:



Read depth relative to ancestor:



Evaluate alignment. The picard-tools function 'Collect Alignment Summary Metrics' yields information about the alignment of reads in a SAM/BAM file, such as the numbers of aligned reads and aligned bases and the mean read length. These metrics can be collected for each sample by the script "picard_align_report.py" and then collated for many samples by the script "compile_picard_report.py." Snippets of the individual sample report and the compiled report are below.

Picard alignment summary metrics:

```
CATEGORY  TOTAL_READS  PF_READS  PCT_PF_READS  PF_READS_ALIGNED  ...
PAIR      2631456      2631456   1             2626143           ...
```

Compiled metrics:

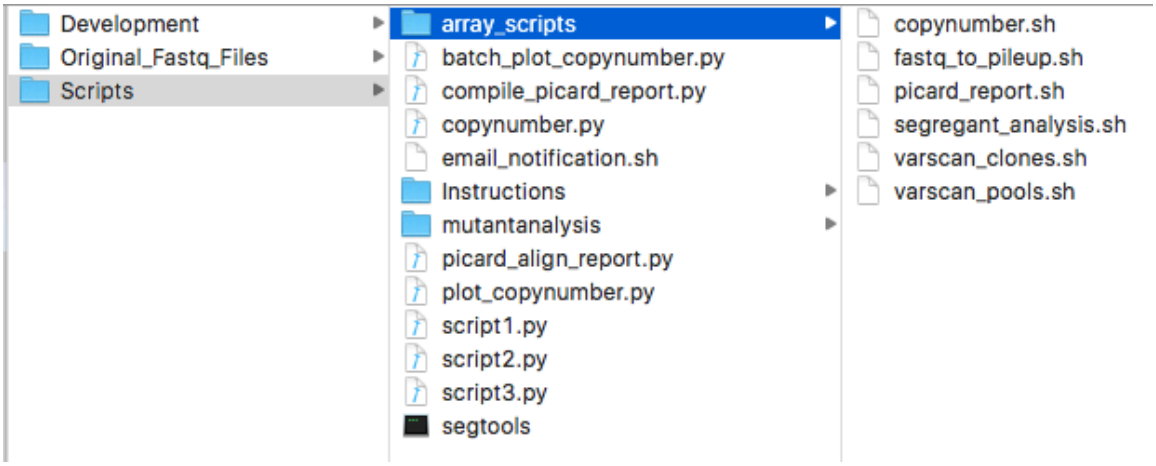
```
sample       total_reads  aligned_reads  percent_reads_aligned  ...
033B         2516654      2411302        0.958138               ...
119_NaCl_HU  6532322      6229601        0.953658               ...
028A         2631456      2626143        0.997981               ...
```

C.2 Executing the pipeline

The user runs three scripts from the command line, corresponding to parts I-III of the analysis described in Section C.1. The user must be logged into the Harvard Odyssey Research Computing cluster, access to the cluster requires registration and must be arranged through Research Computing. The user's main folder on the cluster must contain the appropriate Bash profile, which loads the modules required for executing the pipeline software programs. An example of this Bash profile is available on the lab server in `/murraylab/Sequencing/Scripts/Instructions`. The modules required include: Anaconda (for Python), cutadapt, FastQC, BWA, SAMtools, bamUtil, bamtools, VarScan, ClustalW, and Picard-tools. Also, the environmental variables GATK, VARSCAN, and PICARD should be set to the respective paths for these programs. For

the versions of these programs used in this dissertation's analysis, see Section 4.6.

The complete file tree for running the pipeline is shown in the image below. The programs simply titled "script1.py," "script2.py," and "script3.py" are executed by the user in that order; the "segtools" program and files in the "array_scripts" and "mutantanalysis" directories are called by these programs. The programs "copynumber.py," "plot_copynumber," "batch_plot_copynumber," "picard_align_report.py," and "compile_picard_report.py" are for analyses described in part IV in Section C.1.



All three scripts take as input (1) a directory where the FASTQ, pileup or BAM files are located, (2) an output directory, and (3) a CSV file containing names of the samples to be analyzed, with ancestors, clones and pools listed in separate columns. Each script first checks whether the files it will create already exist and whether the input files it needs are in the directory provided. It then submits this information to the cluster as a separate job for each sample to be run in parallel, using the relevant array script to execute functions either from "segtools" or from "mutantanalysis.py." The required inputs and the format for command-line execution for each script are detailed below.

FASTQ FILES. The original FASTQ files can be placed in the lab server's "Sequencing" directory. No analysis files should be written to this directory. I recommend storing the files in a directory tree that enables other lab members to easily discern the relevant experiment, such as:

```
/murraylab/Sequencing/Original_Fastq_Files/Nichole/2015-01-15/
```

CSV FILE. Execution of the analysis pipeline requires a CSV file containing sample names that uniquely identify the corresponding FASTQ files. In the examples below, there are samples named "006A" and "006B;" an entry of "006" will find the FASTQ files for both samples and result in an error. These sample names should be arranged in two or more columns according to the comparisons to be made and the headings must follow one of the four formats below.

For single clone analysis:

```
Ancestor  Clone
006A      113
006A      114
006B      115
```

For multi-clone analysis:

```
Ancestor  Clone1  Clone2  Clone3  ...
028A      033A    034A    035A    ...
006A      113     114             ...
006B      115     116     117     ...
```

For segregant analysis:

```
Ancestor  Clone  Pool
028A      033A   066
028A      034A   067
028A      035A   068
```

For segregant analysis with multiple pools:

```
Ancestor  Clone   Pool1        Pool2     ...
028A      033A    119_Initial  119_NaCl  ...
028A      034A    152_Initial  152_NaCl  ...
028A      035A    121_Initial  121_NaCl  ...
```

Files could contain just two columns (Ancestor and Clone) or many columns for clones or pools. The analysis to be run – segregant analysis with clone (sawc) or comparing multiple clones (cmc) – is determined from the column headings; e.g., a "Pool" column indicates that segregant analysis should be run. In multi-clone analysis, each clone in a row is compared to the ancestor, and the results for all clones in the row are output into a single set of files. For segregant analysis with multiple pools, each ancestor-clone-pool trio will have its own output.

OUTPUT DIRECTORY. The output directory should be given as a full path to the user's folder on the lab server; i.e., :

```
/murraylab/Users/nmcollin/Sequencing/2015_Seq_Data/
```

SCRIPT1.PY. The first script finds FASTQ files for the samples listed in the CSV file and creates pileup files. All output files are placed into a new folder with the sample name in the given output directory. The script determines whether FASTQ files are present as single or paired-end reads and uses either the 'singlef2p' or the 'pairedf2p' function from the "segtools" script to generate pileup files.

```
input:   python /full/path/to/script1.py
         -i /full/path/to/fastq_files/
         -d /full/path/to/output_directory/
         -f /full/path/to/csv_file.csv

output:  /output_directory/alignment_files/sample/sample.pileup
```

An output pileup file is created for each individual sample listed in the CSV file. The output folder will also contain other files created in pileup process, including the FastQC HTML report, the cutadapt summary file, and the BAM file required for script 3.

SCRIPT2.PY. The second script finds pileup files for the samples listed in the CSV file and creates .snp and .indel files for each ancestor-clone pair and for each pool. All output files are placed into the relevant subdirectory of a new directory "varscan_files." The VarScan files are created by calling the 'varscan_clone' or 'varscan_pool' function from "segtools." These are submitted as two separate array jobs to the cluster: one for ancestor-clone pairs and one for pools. If -d is omitted, the directory is assumed to be the same one specified by -i.

```
input:   python /full/path/to/script2.py
         -i /full/path/to/pileup_files/
         -f /full/path/to/csv_file.csv

         [-d /full/path/to/output_directory/] (optional)

output:  /output_directory/varscan_files/anc_vs_clone/...
           ...anc_vs_clone.snp
           ...anc_vs_clone.indel
         /output_directory/varscan_files/pool/pool.snp
         /output_directory/varscan_files/pool/pool.indel
```

SCRIPT3.PY. The third script finds BAM files created by script 1 and .snp and .indel files created by script 2 and submits these to the "mutantanalysis.py" script to obtain the comparison output for each ancestor-clone-pool or ancestor-clone (-clone...) group. The user can provide separate paths to the BAM and VarScan files, or a single path that encompasses both. If -v and/or -d are omitted, the directories are assumed to be the same one specified by -b. The optional parameters -m and -s are integers between 0-100 that indicate the percent of reads needed to call a mutation (m) or segregant (s). Defaults are m=90 and s=70. Recommended usage for diploid clones is m=35 or lower. The output for each comparison is placed into the relevant subdirectory of a new directory "filename-p-m%-s%" where filename = name of CSV file, p = pipeline (sawc or cmc), m% = mutation percent, and s% = segregation percent.

```
input:   python /full/path/to/script3.py
         -b /full/path/to/bam_files/
         -f /full/path/to/csv_file.csv

         [-v /full/path/to/varscan_files] (optional)
         [-d /full/path/to/output_directory/] (optional)
         [-m mutation percent cutoff] (optional; default:  90)
         [-s segregation percent cutoff] (optional; default:  70)

output:  /output_directory/csv_file-sawc-m90-s70/anc_clone_pool/...
           ...compare.html
           ...tabbed_output_by_gene.txt
           ...tabbed_output_by_mutation.txt
```

Other output files present in this folder are used by the HTML file for viewing aligned reads and FASTA and protein sequence alignments.

EMAIL_NOTIFICATION.SH. When a script executes successfully, it returns a job ID number. The user will be prompted with the option to have an email notification sent once the executed program is completed. To use this option, the user must copy the "email_notification.sh" script from the common directory to their own directory, open the script and change the email address in line 8 to their own. Then, type the following into the command line, with the job ID in the brackets:

```
input:  sbatch --dependency=afterok:[JOB ID #]
          /full/path/to/email_notification.sh
```

To check the status of an executed job, type the following into the command line, substituting the relevant information in the brackets:

```
input:  squeue -u [username] -j[JOB ID #]
```

COPYNUMBER.PY. This script is similar to "script2.py." This script finds pileup files for the samples listed in the csv file and creates .copynumber files for each ancestor-clone pair. All output files are placed into the relevant subdirectory of a new (or existing) directory "varscan_files." The VarScan files are created by the 'varscan_copynumber' function from "segtools."

```
input:   python /full/path/to/copynumber.py
         -i /full/path/to/pileup_files/
         -f /full/path/to/csv_file.csv
         [-d /full/path/to/output_directory/] (optional)

output:  /output_directory/varscan_files/anc_vs_clone/...
           ...anc_vs_clone.copynumber
```

To visualize the results of the copynumber analysis, run "plot_copynumber.py" for one sample or run "batch_plot_copynumber.py" for multiple samples using a CSV file as input. These files create PDF files with plots of copy number for each chromosome.

PLOT_COPYNUMBER.PY. This script takes a single .copynumber file as input and generates three PDF files: one for the ancestor, one for the clone, and one for the clone normalized by the ancestor.

```
input:   python /full/path/to/plot_copynumber.py
         -f /full/path/to/anc_vs_clone.copynumber
         -d /full/path/to/output_directory/

output:  /output_directory/anc_copynumber.pdf
         /output_directory/clone_copynumber.pdf
         /output_directory/anc_vs_clone_copynumber.pdf
```

BATCH_PLOT_COPYNUMBER.PY.This script takes a CSV file as input and generates one PDF file for each individual sample and one for each ancestor–clone comparison, in which the clone's coverage is normalized by the ancestor's coverage. The format of the CSV file should match one of the formats described in part I.

```
input:   python /full/path/to/batch_plot_copynumber.py
         -i /full/path/to/copynumber_files/
         -f /full/path/to/csv_file.csv

         [-o /full/path/to/output_directory/] (optional)
         [-w window size for smoothing] (optional; default:  1000)

output:  /output_directory/anc_copynumber.pdf
         /output_directory/clone_copynumber.pdf
         /output_directory/anc_vs_clone_copynumber.pdf
```

PICARD_ALIGN_REPORT.PY. This script executes the 'Collect Alignment Summary Metrics' function from Picard tools on the BAM file of each sample listed in the CSV file.

```
input:   python /full/path/to/picard_align_report.py
         -d /full/path/to/bam_files/
         -f /full/path/to/csv_file.csv

output:  /full/path/to/bam_files/alignment_files/sample/sample_asm.txt
```

This script could be modified to execute any of the functions in the Picard tools package on sample BAM files by replacing the function name in the "picard_report.sh" array script.

COMPILE_PICARD_REPORT.PY. This script takes the alignment summary files generated by the function above and compiles this information into a single text file. The output file includes computations of coverage levels using a yeast genome size of 12,157,105 bases.

```
input:   python /full/path/to/compile_picard_report.py
         -d /full/path/to/bam_files/
         -f /full/path/to/csv_file.csv

output:  /full/path/to/bam_files/csv_file_alignment_data.txt
```