Final Report: GitGud Project 1 - *Aliivibrio fischeri* Genome Assembly

GitGud: Mason Tatro, Bennett Yunker, Naomi Williamson,Ellie Spahr

***Response to Reviewers***

Proposal review focused on four key concerns; significance, shortcomings of the previous assembler, software choice, and group contributions.

The significance of this study was justified in two parts; (i) generating resources to further the biological understanding of symbioses and (ii) using an assembler with a notably low rate of error (ABySS) to increase genome assembly accuracy. Establishing this system as a model for symbiotic acquisition requires a reliable complete genome for each partner organism. The signalling pathways utilized for symbiotic establishment from the squid side of the relationship are well categorized; the molecular pathways utilized from the bacterial side are relatively unknown in comparison. By having access to a genome for *A. fischeri*, we can more readily predict and elucidate genes integral to symbiosis as well as compare and contrast these elements between other symbiotic bacteria. Our interest in this project does not come from a distinct functional end, but rather to contribute resources so that we may study how symbioses are established mechanistically and from that- how they may have evolved.

Drawbacks of the 2019 assembly software choice (SPAdes) were omitted as none were notable. SPAdes is a new hybrid assembler that performs very well for assembly of small prokaryotic genomes. Due to our limitations by only having access to those assemblers available in the bioconda environment, we elected to choose a more accurate assembler (ABySS) as determined by the GAGE-B assessment. Ultimately, using already published data through NCBI’s SRA database, we selected our data based on organism and not the previous assembler used. Their choice was very appropriate for the task so we sought to improve upon it in the only way we could find.

The Read Quality software was amended; initial quality control was conducted by FastQC. Trimming software used was Trimmomatic.

Roles for each group member were vague due to misinterpretations on proposal and software requirements (identifying pipeline steps vs specific packages). Specific roles were assigned as follows; Ellie and Naomi ran quality control, trimmed the data, and evaluated results through secondary QC. Ben and Mason lead the effort to write code and optimize assembly using ABySS. Ellie, Naomi, and Mason contributed to the final report document.

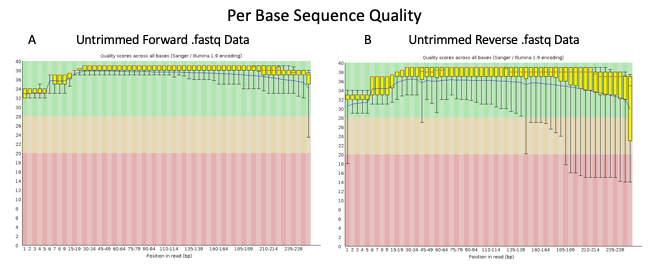
***Results***

*Data Retrieval*

*Aliivibrio fischeri* fastq data was retrieved from the NCBI SRA database (SRA: SRR8647324, Experiment: SRX5445104) and copied to the Spruce Knob Supercomputer for analysis.

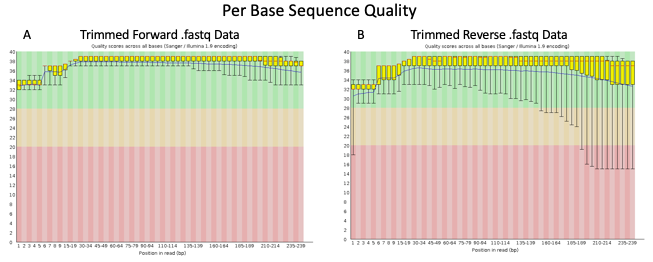
*Quality Control and Trimming*

Quality control was conducted using FastQC (ver 0.11.7) in Bioconda. Trimming was conducted through the bioconda package Trimmomatic (ver 0.38). Initial quality control was run on our paired-end data through a single file. The first trimming submission attempted to remove small reads below 200nt, but was terminated as the paired end data was interleaved into a single file and violated the ‘PE’ argument.



**Figure 1. FastQC output for initial Quality Control.** Per Base Sequence Quality graphs from pre-trimming QC showed a small proportion of data with poor base quality in the 3’ ends of reads for (A) forward and (B) reverse sequences.

FastQC was rerun on the separated read files, but only output results for the forward sequences. We trimmed to 240 bases and removed reads under 200 nt. Trimming was able to discard truncated sequences from a partial reverse fastq file (file upload had been disrupted) and post-trimming fastqc output both forward and reverse results. Reverse files were removed from scratch and re-uploaded successfully; FastQC for the amended reverse file was successful. FastQC results for per base sequence quality of forward and reverse fastq files is represented in Figure 1.

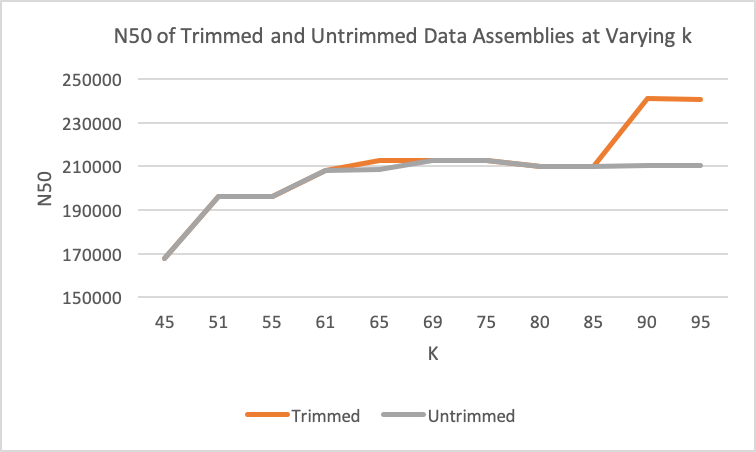


**Figure 2. FastQC output from Trimmed Data**. Per Base Sequence Quality from trimmed reads reduced the 3’ read noise in (A) forward reads and (B) reverse reads.

Trimming was conducted a final time with corrected files (CROP:240, MINLEN:200), along with one final post-trimming FastQC run. Interspersed low quality base calls that were observed in the untrimmed reverse data were discarded with the short read sequences (Fig 2B). A small proportion of low-quality 3’ bases were present, but average base sequence quality remained high. No sequences were flagged as poor quality in the pre- or post-trimming FastQC runs. During trimming, number of reads was reduced from 1,885,459 to 594,230 in each file; though these discarded reads were small in size (30-199 nt) and likely contributed a low proportion of total data, we attempted to minimize coverage loss by trimming conservatively.

*Data Processing*

Our first successful ABySS (ver 2.1.0) assembly ran with k=79. To construct the highest quality assembly, data was run with multiple k values from 50-75 in increments of 5. This was then expanded to include both trimmed and untrimmed *A. fischeri* data from k values of 45 to 95.



**Figure 3, Trimmed and Untrimmed Data Assembly at Varying *k*.** Data was run before and after trimming at k-values from 45-90 to optimize assembly parameters. This graph illustrates the change in N50 at varying k values, between data sets.

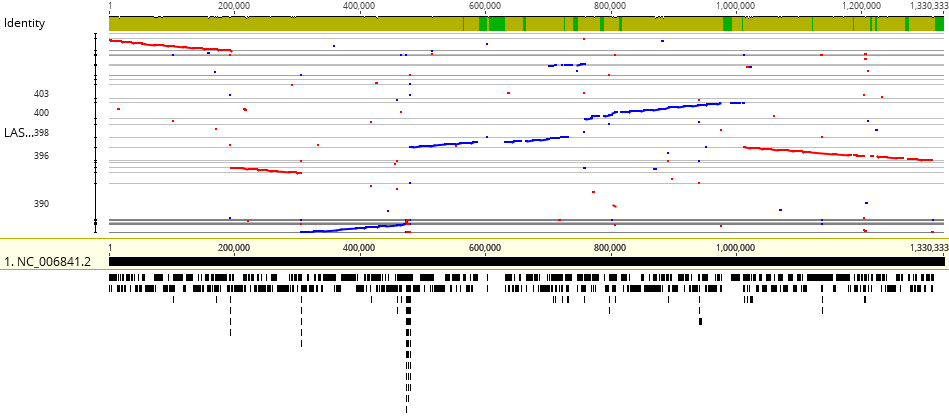
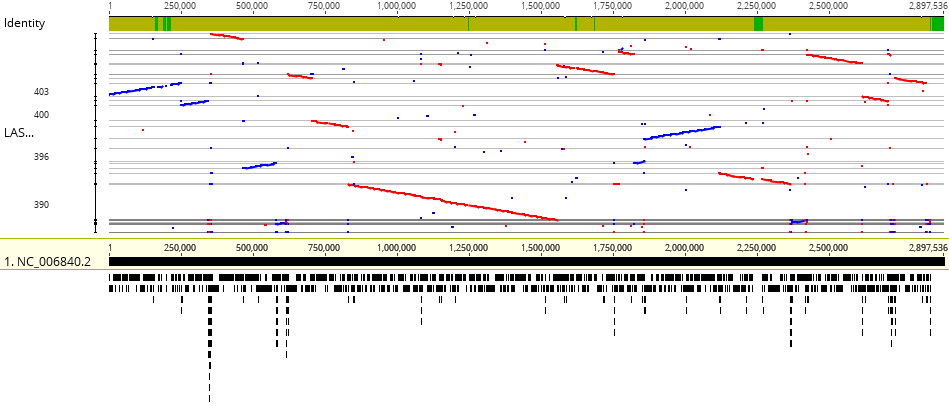
Assembly outputs for varying *k*-values were evaluated by N50 and n values, though L50 was also considered when optimizing k-mers (see Discussion). Figure 3 illustrates the change in N50 across our assembly conditions. The largest N50 value was achieved for trimmed data at a *k* of 90. This assembly also contained the lowest n across all assemblies (contigs: n=155 at k=90). To complete k-mer optimization, we closed the 90-95 k window by assembling with k-mers of 91, 92, 93, and 94 to capture the optimal k. When comparing the final results against those generated from k=90, k=92 showed the highest N50, the lowest n (n=151), as well as a low L50, suggesting more data was captured by fewer contigs and, in turn, larger coverage contigs than seen at other *k*s (see Table 1).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | n:500 | L50 | min | N80 | N50 | N20 | E-size | max | sum | name |  |
| 228 | 62 | 9 | 503 | 95508 | 191253 | 261215 | 195682 | 419465 | 4263273 | vfisch92-unitigs.fa | |
| 151 | 35 | 6 | 540 | 114719 | 240931 | 336011 | 301072 | 765530 | 4278329 | vfisch92-contigs.fa | |
| 143 | 27 | 4 | 540 | 125849 | 290664 | 617643 | 375828 | 765530 | 4277803 | vfisch92-scaffolds.fa | |

**Table 1. ABySS statistics for genome assembly using k=92.** The complete output statistics for the “best” genome assembly can be seen above, as generated from a k-mer of 92.

*Post-Processing*

The proposed post-assembly quality control software, QUAST, was not available in bioconda on Spruce. Instead, we elected to further assess the completeness of our assembly output by aligning it to a NCBI’s reference *A. fischeri* genome (ES114) using the LASTZ plug-in for Geneious.



**Figure 4. Alignment of the final assembly to a reference *A. fischeri* genome (ES114).** The contigs generated from our assembly were aligned to the current reference *A. fischeri* genome as retrieved from NCBI. The *A. fischeri* genome consists of two circular chromosomes (2.8 Mbp - top, 1.3Mbp - bottom). Contig sequences from the final assembly were aligned to both chromosomes. Identity is represented at the top of the graph; alignment of contigs is mapped along the bottom.

***Discussion***

*Expected versus Actual Outcomes*

To optimize k-mer values we evaluated ABySS output for contigs (paired sequences with scaffolding over covered gaps with no repeats) and scaffolds (paired end sequences with coverage gaps and repeats), but not unitigs. We first assembled at k-values of 50-75 in steps of five in an effort to test wall time and observe preliminary trends in our ABySS statistics. In commonly cited metrics, such as contig number, N50, and L50, it appeared that our assemblies were improving at larger k values. It quickly became clear that our ideal k was likely greater than 75.

Due to the reduction in the number of reads between trimmed and untrimmed sets, we alo assembled both data sets in our next ABySS iteration to test the contribution of small reads to final assembly size. There was little additional contribution from untrimmed data; assembly sum remained very similar between assemblies of equal k. By graphing the N50 of contigs across k, we were able to track the contribution of k to the read lengths contributing to half of the total assembly. Typically, a higher N50 is preferred because this denotes the minimum length of contigs comprising 50% of the assembly and suggests the assembled genome is more complete (often compared to a weighted median value). We observed n (contig number) as a function of k, as well, since fewer contigs means less gaps between regions of genomic coverage. Our initial sweep suggested trimmed and untrimmed data assemblies were comparable in k values from 45-85, but k values above 85 did not improve the quality of untrimmed assemblies. For our trimmed reads, k=90 appeared to be close to our ideal k as n and L50 were lower than assemblies of k=85 and k=95, but the N50 was at its highest value. The decrease of N50 from k=90 to k=95 suggested a loss of assembly efficiency at k=95 and more fractionated contigs. We increased optimization by assembling within the 90-95 k window; The 92 k-mer size gave an assembly with the lowest number of contigs and scaffolds. This k-mer value also gave the highest N50 value, with low contig and scaffold L50. We therefore found a k-mer of 92 to be optimal when assembling our trimmed data sets and proceeded with post-processing.

While the total number of contigs in our *de novo* assembly seems large (151 contigs), looking at the data in Geneious showed a very large number of very small reads; a small number of contigs counted for a majority of our coverage (as corroborated by our L50 as seen in Table 1). By mapping the assembled contigs to the reference genome, we were able to easily visualize coverage. We had anticipated the possibility of many small contigs inflating our n, but had also expected more continuous genome coverage. We were, however, pleased with the alignment spread of our contigs across both linearized chromosomes. Despite many remaining gaps in our data, its ability to span most of the reference data suggests an adequate assembly for the resources and time involved. In order to rectify these gaps in coverage, any further *de novo* assembly attempts would require the inclusion of more raw data or the use of another assembler that may perform better for a small prokaryotic genome such as MaSuRCA.

*Lessons Learned and Troubleshooting*

We encountered several complications during our assembly process that required varying degrees of troubleshooting, but also provided valuable learning experiences. Early errors occurred in our pipeline due to writing shell scripts on a Windows environment and submitting them to Spruce, resulting in line ending incompatibility. Once these were changed to be compatible in a Unix environment, the scripts ran as normal. The next error we encountered was repeated early job termination for Trimmomatic. When we first downloaded our paired end data from the NCBI SRA database, we received one file. This didn’t disrupt our ability to run FastQC, but had halted Trimmomatic by attempting to submit a single g-zipped fastq file with a ‘PE’ argument; de-interleaving the data and specifying two input files allowed our script to run without error.

After resolving the Trimmomatic issue and retrieving the separated data files, we noticed an inability to generate initial fastqc data for the reverse read files when compared to our post-trimming fastqc run. Re-downloading files from $SCRATCH showed that the reverse sequence file was half the size of the forward set; the truncated file set was removed from scratch and re-uploaded. When initially reuploading the two new fastq data files off-campus, Spruce had dropped the connection and truncated the reverse fastq.gz file, despite reading as complete. Removing and reuploading the reverse data fixed missing or incomplete outputs.

When first running ABySS, jobs initially failed due to wall-time errors; PBS -q argument was changed to comm-mmem-day and wall time increased to resolve the issue. We first attempted to run multiple jobs in one script and use the -C argument to specify output to k-specific directories within scratch. This seemed to create an error within the input file path and terminated jobs with an error saying the input fastq files were not found, despite that portion of code being unchanged. To combat this, we removed the -C argument and manually moved the output files to their respective directories after job completion.

Most of our troubleshooting came from errors that weren’t readily visible in the code, but hidden line endings or incomplete files that prevented us from generating the intended output. Moving through the *de novo* assembly process highlighted the necessity of checking the less obvious variables and not just the coding strings.

*Pipeline code*

# FastQC run

# vfischqc.sh contents:

#

#! /bin/bash

module load genomics/bioconda

source /shared/software/miniconda3/etc/profile.d/conda.sh

conda activate tpd0001

cd $SCRATCH

fastqc vfisch1.fastq.gz vfisch2.fastq.gz

conda deactivate

# Trimming with de-interleaved data

# Followed by post-trim QC

#

#! /bin/bash

module load genomics/bioconda

source /shared/software/miniconda3/etc/profile.d/conda.sh

conda activate tpd0001

trimmomatic PE -threads 6 -trimlog VfTrimm vfisch1.fastq.gz vfisch2.fastq.gz -baseout vfisch\_trim2.fastq.gz CROP:240 MINLEN:200

fastqc vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz

conda deactivate

# ABySS assembly with a k-mer length of 79nt

#

#! /bin/bash

module load genomics/bioconda

source /shared/software/miniconda3/etc/profile.d/conda.sh

conda activate tpd0001

abyss-pe name=vfisch k=79 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

conda deactivate

# ABySS Assembly from k=50-75 in increments of five.

#

#! /bin/bash

module load genomics/bioconda

source /shared/software/miniconda3/etc/profile.d/conda.sh

conda activate tpd0001

cd $SCRATCH

abyss-pe name=vfisch75 k=75 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch70 k=70 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch65 k=65 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch60 k=60 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch55 k=55 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch50 k=50 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

conda deactivate

# ABySS Assembly to capture steps from 46-95, trimmed and untrimmed data.

#

#! /bin/bash

module load genomics/bioconda

source /shared/software/miniconda3/etc/profile.d/conda.sh

conda activate tpd0001

cd $SCRATCH

abyss-pe name=vfischtrim45 k=45 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim51 k=51 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim55 k=55 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim61 k=61 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim65 k=65 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim69 k=69 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim75 k=75 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim80 k=80 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim85 k=85 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim90 k=90 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischtrim95 k=95 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfischuntrim45 k=45 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim51 k=51 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim55 k=55 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim61 k=61 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim65 k=65 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim69 k=69 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim75 k=75 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim80 k=80 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim85 k=85 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim90 k=90 in='vfisch1.fastq.gz vfisch2.fastq.gz'

abyss-pe name=vfischuntrim95 k=95 in='vfisch1.fastq.gz vfisch2.fastq.gz'

conda deactivate

# Assembling at k=91, 92, 93, and 94

##! /bin/bash

module load genomics/bioconda

source /shared/software/miniconda3/etc/profile.d/conda.sh

conda activate tpd0001

cd $SCRATCH

abyss-pe name=vfisch91 k=91 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch92 k=92 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch93 k=93 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

abyss-pe name=vfisch94 k=94 in='vfisch\_trim2\_1P.fastq.gz vfisch\_trim2\_2P.fastq.gz'

conda deactivate

*Supplemental Data*

Individual ABySS statistics, as well as the complete initial QC reports, can be found at /wvu\_gda/gitgud and /wvu\_gda/gitgud/trimmed\_untrimmed in the course github repository.