Final Project Report

**HOW TO RUN PIPELINE ON THE HPC:**

1. Extract project folder. There should be three sub-folders, two .job files, and one .sh file. The folders are used to organize the project and access files in a simple way.

|  |  |
| --- | --- |
| FILE/DIRECTORY NAME | PURPOSE |
| assembly/ | Hold split .faa files for job submission |
| programs/ | Contains PERL scripts for downloading/splitting genome and analyzing BLAST results |
| results/ | Folder to output BLAST results and the final analysis |
| analyzeResults.job | Script for HPC. Runs PERL script in 1 slot to process BLAST results |
| runblast.job | Script for HPC. Runs BLAST on targeted file and outputs to results. |
| main.sh | Script to run on login node of HPC to tie the pipeline together. |

Files and project structure will be discussed in the Final Report

1. Transfer project folder to HPC. Navigate to folder above extracted project folder. Then, use scp as follows:

scp -P 40 -r ./finalProject/ *username*@neon.hpc.uiowa.edu:

1. On the HPC, set CD to finalProject directory:

cd ./finalProject

1. On the HPC, run main.sh -> will download the genome, run BLAST jobs, and analyze the output:

sh main.sh

Pipeline Diagram:

SETUP – HPC LOGIN NODE:

Download genome, split into files with 10 entries/genes each

BLAST JOBS:

Takes in split files. Outputs BLASTp result files to ./results

ANALYZE JOB:

Iterates and filters BLAST results. Sends top filtered hit to a collective annotation file

**CODE OVERVIEW**

Overall, the pipeline was written to be modular. The user can easily change the target genome, number of jobs/split files, blast parameters/results, and the output locations.

**main.sh**

There are two variables in main.sh that the user can change: file and entriesPerFile. “file” is a ftp address to a genome. “entriesPerFile” is the number of files the genome entries will be distributed into; it is also the number of BLAST jobs submitted to the HPC.

**fileBreakdown.pl**

Main.sh then calls a PERL script, fileBreakdown.pl, on line 15 and passes the variables as parameters. In fileBreakdown.pl, the script uses wget to download the genome into the ./programs directory and unzips it with gunzip. If the program fails to download/open the file correctly, it fails. The PERL script then goes through the file and separates the .faa file into X entries per file. The split files are saved to ./assembly and named as splitFileN, where N is an index between 1 and X.

After this, the main.sh checks the file count in ./assembly. If there are not the expected number of files, the script terminates before submitting a job to the HPC. Otherwise, main.sh submits an arrayjob with X jobs. These jobs use the runblast.job script.

**runblast.job**

Sets up qsub with 4 slots on the DK queue. Calls blastp with refseq\_protein database on a splitFileN file. Outputs to ./results named resSplitN with 20 alignments and a custom format (details later on).

The main.sh then submits a hold job, analyzeResults.job, that will not execute until all X blast jobs are completed.

**analyzeResults.job**

Deletes the existing result file if it exists. Then, the script iterates over all BLAST result files (resSplitN files) and calls a PERL script on them (analyzeFile.pl). There are 3 parameters: the BLAST result file, number of entries per file, and result file. The user can change the result file name if they want to save the analysis to a unique location. Script checks parameters for validity and terminates if they are wrong. The scripts goes through the blast result file in chunks pertaining to a unique entry. It filters any result with the same organism as the top result (which is assumed to be the query; however, this behavior can be changed in the code for future pipeline runs where the query does not exist in the database). The code sorts the results by a BLAST flag (percent positive in this case) and adds the top hit with the necessary details to the output file.

**DETAILS**

Genome used: Mycoplasma genitalium G37

<https://www.ncbi.nlm.nih.gov/assembly/GCF_000027325.1#/st>

<ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/027/325/GCF_000027325.1_ASM273>v1

The selected genome was provided as a recommended genome to download and annotate due to it being a small genome to analyze. A user can edit the link in the main script to annotate any genome in a database. The selected genome was filtered out in the annotation process to simulate annotating a new genome/sequence.

Various nodes on the HPC have different numbers of cores and memory available. The number of files the genome is broken down into is the number of jobs submitted to Neon. This feature was implemented to allow job control for efficient processing. The file is downloaded and split with a PERL script. The genome used in this project was downloaded from the refseq database. The download was the .faa.gz version -> can uncompress with gunzip and have a predictable layout from the .faa format. The .faa layout allows the PERL script to identify, count, and split the entries into separate files. Having the amino acid version is less memory and processing compared to nucleotide .faa format.

**Modularity:** The arguments in various stages of the pipeline allow the code to be modular and customizable. This would allow running the pipeline against any genome online or uploaded. The various options allow for control over the BLAST parameters, output, job submission, ect. Later, I took use of the modularity to run specific files and BLAST queries. The script checks over the various stages of the process. It makes sure the genome was downloaded and split into files. If one of the checks fails, the entire pipeline is stopped -> jobs are not submitted to HPC.

**Parallelization strategy:** The genome file was split into files of 10 sequences each to take use of running BLAST in parallel, and to narrow down where specific annotations are from. The actual parallelization was implemented using an array job on the Neon HPC. The main code counts the number files the genome was split into, and then uses that variable as the number of jobs in the arrayjob. I did this to allow any genome to work and run with arrayjobs. Hardcoding every value in the code makes adjustments difficult and prone to error. The analysis is done after all arrayjobs are completed. This is done using a flag in the qsub command. The analysis uses one slot since it is fast processing. The PERL script used for the analysis can be adjusted to filter by variables from the BLAST results. Also, the analysis script can take in one, a range, or all BLAST file results and output to a specific file, so it can be called multiple times and return with the selected filter.

**BLASTp:** Blastp was used because of the genome file format. Maybe in a future version of the pipeline, it can detect the file/sequence format and run the appropriate BLAST program. The refseq\_protein database was used to eliminate partial proteins results and to acquire known proteins with annotations. The BLAST results used a custom output format to output results. The custom format is “Q Sequence ID, S Sequence ID, S title (annotation), mismatch count, evalue, percent identical, percent positive”. Not every output variable is included in the final annotation file, but are there if more info or filtering is desired.

The number of alignments was selected to be 20 (at first, then was extended to 60). Reasoning was: using the refseq database, if there are not any BLAST hits from another organism in the top 20, then the gene is unique to the organism. On the flip side, if this pipeline ran with a well-documented genome (human) then there might not be any from another organism for a while (unless the query organism was filtered out during the BLAST jobs).

Number of threads was selected to be 4 for BLAST. A BLAST run with one thread would not work with one slot on the DK queue in the Neon HPC. Using four slots would work. Using four threads would speed up the BLAST process and use the available resources efficiently.

**Annotations:** Annotation format: 1 annotation per line, tab delimited. First column is database accession number, second column is percent positive, and third column is the annotation. For the final annotation, I used percent positive to select an annotation for the query sequence. Percent positive is like percent identity, except it includes similar amino acid pairs. The function and structure of a protein relies on the sequence. If two sequences are closely related, then there is a better chance their functions are the same.

Results from first pass: 483 annotations. All annotations are from another organism in the database. There were no excessive annotations from any file (>10 annotations per 10 entries), but lower than expected in several. I noticed the BLAST results that had no annotations were ones where every BLAST hit was a sequence or hypothetical sequence of the query, so my filtering algorithm would not return anything. I had the limit of results set to 20, and thought there would be more hits/annotations if the limit was increased. To test this, I adjusted my filter to 60 hits and reran BLAST on the files that had less than expected annotations. This increased the annotation count to 491. Looking at the results, I could tell increasing the result count might not help much more. The reason the annotation count is below 515 is because of two factors: BLAST hit count and the BLAST organism of the hit. Some entries generated less than twenty results, so increasing the hit limit would not help. Those results were all hypothetical and were of the query genome (mycoplasma genitalium), so the results were filtered out while creating the annotation file. So, I adjusted the threshold of the filter and increased the annotation count to 495. 20 genes are unaccounted for, probably due to the limitations of the filtering algorithm.

**NOTES:**

Project Dir

Contains: file to run

programs

Contains:

Perl scripts

assembly

Contains:

Folders and split files

Results from BLAST

BLAST executables and reseq\_protein database: /Shared/class-BME5320-dkristensen/

Guideline of steps in pipeline:

Setup up file (at head/login node)

* Download genome from command line. Uncompress to directory \_\_\_\_\_\_\_\_\_\_\_\_\_
* Check downloaded file for correct number of proteins
* Split file into separate folders/files

Each job/folder will

* For each .faa folder/file (from above) submit a job? qsub to a script file

Run blast on the .faa file. Arrayjob

* + - Tab output
  + Analyze results