Genome Majority Vote (GMV), Version 0.1

Manual

Sindhu Raghavan
Michael E. Wall

Initial version date: 7 Aug 2009
Present version date: 18 May 2011

1. Software/Platform requirements

The following software has to be installed to be able to run the pipeline successfully:

a) **Mac/Linux –** This software has been developed and tested on Mac and Linux. It has not been tested on Windows. Make sure you are running the pipeline either on Mac or Linux.

b) **Perl** - Perl 5.8 was used to develop and test the software. Perl 5.8 or higher can be used to run the software.

c) **Java –** JDK 1.6 was used to develop and test the software. Make sure JDK 1.6 or higher is installed. Do not use the java compiler/interpreter that comes with the Linux installation.

d) **Prodigal –** Prodigal is a bacterial gene prediction software and can be downloaded from the following website http://code.google.com/p/prodigal/. We originally used Prodigal version 1.10 to develop and test the pipeline. **The present version GMV 0.1 supports Prodigal versions 2.00 through 2.50.** After downloading Prodigal, compile the source and test it on a sample data set before running the pipeline. The instructions for compiling and running Prodigal can be found on the Prodigal website. ***Since Prodigal is developed in C, the source has to be compiled from the scratch if you want to run the pipeline on different machines.***

e) **BLAST –** BLAST is a local alignment search tool and can be downloaded from the following website – http://www.ncbi.nlm.nih.gov/BLAST/download.shtml. Download the correct version of BLAST depending on the platform on which the pipeline will be run. We have used BLAST 2.2.20 to develop and test the pipeline. Instructions for installing and testing BLAST can be found on the BLAST website. Make sure BLAST works fine before running the pipeline. Since BLAST is a computationally intensive program, one might want to run the steps that involve BLAST on a powerful server machine. Also, since BLAST is parallelized, one can run BLAST on multiple processors if a multi core machine is available.

Instructions on running different steps of the pipeline on different machines will be given later in this document.

f) **Muscle –** Muscle is a multiple alignment software and can be downloaded from the following website – http://www.drive5.com/muscle/download3.6.html. We have used Muscle 3.7 to develop and test the pipeline. Instructions for installing and running Muscle can be found on the Muscle web page. Make sure Muscle works fine before running the pipeline.

2. Steps for installing the software

1. Download the code at http://code.google.com/p/gmv/ and unpack the .tgz archive (tar –xzf gmv-0.1.tgz) in the directory where you want the root directory gmv-0.1 to reside.

2. The gmv-0.1 directory has the following structure:
    1. code/bin – contains the java class files
    2. code/gov – contains the java source files
    3. code/scripts – contains the perl scripts
    4. code/example/input – contains a sample data set
    5. code/example/output – contains sample output
    6. code/compile.sh – script to compile the java source
    7. doc/ – contains the manual (this file)
    8. input/ - placeholder for input sequences
    9. output/ - placeholder for output
    10. work/ - placeholder for working directory
    11. 00LICENSE – license file
    12. 00README – readme file

3. Compiling the java source files. ***(This is an optional step and the source does not have to be compiled. Only if the user made changes to the java code, compiling is necessary).***
    1. Change to code/.
    2. Edit the path to the java compiler in compile.sh. ***The current version of the software has been developed and tested on Java 1.6 on both Linux and Mac. Make sure the correct version of Java is installed.***
    3. Execute compile.sh

4. Edit the file code/scripts/config
    1. SCRIPTS_DIR – this should point to code/scripts.
    2. MUSCLE_DIR – this should point to the directory where the muscle executable exists.
    3. PRODIGAL_DIR – this should point to the directory where the prodigal executable exists.
    4. BLAST_DIR – this should point to the directory where the BLAST executable exists.5. PERL_BIN – this should point to the directory where the perl executable exists.
    6. JAVA_BIN_DIR – this should point to the directory where the java interpreter executable exists.
    7. JAVA_CODE_DIR – this should point to code/.

Config file can be saved anywhere as long as the full path to it is specified while running the pipeline.

3. Running the pipeline

Change to code/scripts directory and run start.pl for options. You will see the following:

Usage :: perl start.pl

-i Input directory contains all the sequence files in fasta format. <REQUIRED>

-o Output directory contains the results. <REQUIRED>

-w Working directory cotains all the intermediate results. <REQUIRED>

-f 1 to remove all intermediate files and 0 to retain intermediate files. Default 0.

-c Complete path to the configuration file. <REQUIRED>

-v Logging level. 1 is ERROR, 2 is WARNING, 3 is INFO. Default is 1.

-l Log file with complete path. Default - Log messages will be written to standard output

-n Number of processors on the machine on which BLAST could be run. Default 1.

-a 1 to turn on filtering algorithm and 0 to turn off filtering algorithm. Default 0.

-t Threshold for filtering algorithm - Number of consistent prodigal starts. Default is majority.

-b Begin step. See below for step numbers. If only the start step is specified, the program will run from the start step till the end. Default is run all steps.

-e End step. See below for step numbers. If only the end step is specified, the program will run from the beginning till the end step. Default is run all steps.

Step Numbers

1 Create working directory and train and run Prodigal.

2 Process Prodigal files to obtain all possible start sites for all genes.

3 Create BLAST database.

4 Run BLAST.

5 Compute percent identity.

6 Get orthologous sets of genes.

7 Generate input files for alignments.

8 Get muscle alignments.

9 Prediction of start sites.

Options:

1. **-i** Complete path to the input directory containing the genome sequences in the **fasta format**. There has to be exactly one fasta file per genome. All the sequences (complete genome sequence and plasmids if any) for a genome should be in one fasta file. Make sure that the headers for the sequences in the fasta file do not have any spaces in them and are not very long. Restrict the length of the headers in the fasta file to 10-15 characters. It is advisable to use the NC numbers for the headers as they are short and unique. A sample data set is provided in code/example/input, with output in code/example/output, and intermediate and log files in code/example/work.

2. **-o** Complete path to the output directory where the results will be saved. The output directory has the following structure:

    1. **alignments** – this directory contains the alignments in the fasta format for all orthologous sets of genes. The headers for the sequences in the fasta files have internal names of the form <genome_id>_<segment_id>_<gene number>. The mapping for the <genome_id>_<segment_id> can be found in <results_dir>/mappings/segment-name-mapping.txt.

    2. **Feature-files** - this directory contains the feature files that can be used to visualize the alignment files (fasta files) in Jalview software. Based on the information in the feature files, different start codons are colored with different colors. Red color indicates a corrected start codon. Blue color indicates a Prodigal start and green color indicates a start proposed by Prodigal.

    3. **format -** this directory contains the output files in different formats. For the moment, we support only GenBank format. The output files in GenBank format can be found in format/GBK directory. The GBK directory has a directory for each genome in the input data set. Each of these directories consists of the cds files and the protein files.

    4. **input-sequences –** this directory contains the input sequences. If the input fasta file has multiple sequences, it is split into multiple fasta files having a single sequence.

    5. **mappings –** this directory contains mapping files – genome-name-mapping.txt and segment-name-mapping.txt. Genome-name-mapping.txt has the mapping for the genome

names and genome ids. Segment-name-mapping.txt has the mapping for <genome_id>_<segment_id>.

6. **warnings –** this directory contains the files gene-pairs-tied.txt and inconsistent-gene-pairs.txt. Gene-pairs-tied.txt contains all the different pairs of genes that have the same normalized sequence identity. As a result, these genes were rejected while generating orthologous sets of genes. Inconsistent-gene-pairs.txt contains genes for which the best match in its genome is not itself. This file is usually empty.

7. **blast.tar.bz2 –** BLAST results compressed as a .tar.bz2 file. To decompress it, use the following command – tar -xjf blast.tar.bz2

8. **gene-predictions.txt –** this file contains all the gene predictions.

9. **log.txt** – if the user has specified a log file while running pipeline, a copy of the log file will be created in the results/output directory.

10. **orthologs-stats.txt –** this file contains the ortholog sets and several statistics like minimum percent id, maximum percent id, average percent id etc.

11. **summary.txt** - this file contains the high level summary of the results.

12. **time-log –** this file contains the time taken by each step in the pipeline.

3. **-w** Complete path to the working directory. Working directory is a directory where the intermediate results will be saved. The working directory will be removed when the pipeline finishes if -f is set to 1. Default option is to keep the working directory. The working directory has the following structure:

    1. **alignments** – this directory contains the input fasta files for the alignment program, alignment files in fasta format and the feature files for Jalview program.

    2. **blast** - this directory contains the blast database files, the query files and the results files.

    3. **fasta -** this directory contains one sub directory for each genome in the input data set. Each sub directory contains the input sequences for the genome, the prodigal training files and the prodigal gene prediction results. prodigal_prediction directory has the symbolic link to the input files used in the prediction step. prodigal_results directory has the results from prodigal. prodigal_training has the full genome file as well as the training file generated by prodigal during the training step. Also, each sub directory has segment-name-mapping.txt that gives the mapping between the internal id used as the header in the fasta files and the actual header present in the input files.

    4. **ortholog_sets -** this directory contains all the ortholog sets identified, one file per ortholog set. The name of the file is the name of the reference gene in the ortholog set. Each ortholog set file has the names of genes in the ortholog set.

    5. **prediction** – this directory has files generated during the prediction step of the pipeline. Gene-predictions.txt has all the gene predictions, ortholog-stats has various statistics for the

ortholog sets, summary.txt has the high level summary of the results, and predicted-genes-features.txt has information necessary for future use.

6. **prodigal_genes** – this directory has the list of prodigal genes generated for each genome in the input data set.

7. **std_identity** – this directory has the standard identity values for the different pairs of genes.

8. **warnings** - this directory contains the files gene-pairs-tied.txt and inconsistent-gene-pairs.txt. Gene-pairs-tied.txt contains all the different pairs of genes that have the same normalized sequence identity. As a result, these genes were rejected while generating orthologous sets of genes. Inconsistent-gene-pairs.txt contains genes for which the best match in its genome is not itself. This file is usually empty.

9. **all-genes-startsites.txt** – this file contains the prodigal starts and the corresponding scores for all genes predicted by prodigal. Each line is of the form <gene_id>, <prodigal_start>,<prodigal_end>,<direction>,<possible_starts>,<scores>

10. **all-genes.txt** - this file contains the list of all genes predicted by Prodigal.

11. **best-gene-pairs-percent-id.txt** - this file contains the best gene pairs along with their percent id.

12. **best-gene-pairs.txt** - this file contains all the best gene pairs.

13. **genome-name-mapping.txt** – this file contains the mapping for genome names and genome ids.

14. **msa-start-end-pos.txt** - this file contains start and end position of the sequence given as input to the MSA program. Each line is of the form <gene_id>,<start pos>,<end pos>,<direction>.

15. **proteins-length.txt** - this file contains the length of proteins for all the proteins generated by Prodigal. Each line has the following format : <gene_id>,<prodigal_start_pos>,<prodigal_end_pos>,<direction>,<length of the corresponding protein>

16. **selected-genes-rel-startsites.txt** – this file contains the start pos, end pos, direction, all possible starts, corresponding scores, start and end of the segment to the MSA program for every gene.

17. **time-log -** this file contains the time taken by each step in the pipeline.

4. **-f** flag to remove/retain working directory. If 1, working directory is removed. Default is 0.

5. **-c** complete path to the config file. A sample config file can be found in /code/scripts/config.

6. **-v** Logging level. If 1, only error messages are logged. If 2, error messages, warnings and

information about the different steps in the pipeline is logged. If 3, debug messages are also logged. Logging level 2 is the optimum logging level.

7. **-l** Complete path to the log file. If no log file is specified, log messages will be written to standard output.

8. **-n** Number of processors on which BLAST can be run. If a multi core machine is not available, then BLAST will be run on a single processor.

9. **-a** 1 is for the filter based algorithm and 0 is for the score based algorithm. Default is 0.

    1. **Filtering based algorithm** - In this algorithm, the set of aligned start sites in which the number of Prodigal starts exceeds the specified threshold is chosen as the preferred set of starts. Ties are broken based on the average score.
    2. **Score based algorithm** - In this algorithm, the set of aligned start sites that has the best average score is chosen as the preferred set of starts. Ties are broken based on the number of Prodigal starts.

10. **-t** Threshold for the filtering based algorithm . Threshold should be any number between 0 and number of genomes in the input data set. A threshold of 0 is equivalent to score based algorithm. Default is majority as this has shown to give the most optimum results in our experiments. In case of an invalid threshold entry, default is assumed.

11. **-b** Begin step in the pipeline. This is used when the user wants to run the pipeline in multiple stages. Default is step 1.

12. **-e** End step in the pipeline. This is used when the user wants to run the pipeline in multiple stages. Default is the last step.

4. Running the pipeline in single stage (one shot)

The following is used to run the pipeline in a single shot:

perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold>

Examples of parameters:
<input_dir> - /home/sindhu/data/input
<output_dir> - /home/sindhu/data/output
<working_dir> - /home/sindhu/data/working
<log file> - /home/sindhu/data/working/log.txt
<config file> - /home/sindhu/code/scripts/config
<algorithm> - 1
<threshold> - 3

The above command will look as below when the actual parameters are given:

perl start.pl -i /home/sindhu/data/input -o /home/sindhu/data/output

-w /home/sindhu/data/working -v 2 -l /home/sindhu/data/working/log.txt
-c /home/sindhu/code/scripts/config -a 1 -t 3

5. Running the pipeline in multiple stages

It is possible to run the pipeline in multiple stages. Consider a scenario in which a user does not have a powerful desktop computer, but has access to powerful server machines. Now, the user might want to run few steps of the pipeline on the server machine and the remaining steps on his desktop computer. Let us assume that the user would like to run the steps involving BLAST and Muscle on the server machine. In order to achieve this, do the following:

a) Run the first two steps on the desktop machine.

perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold> -b 1 -e 2

b) Copy the input, output, and working directories to the server machine. Run steps 3 and 4 that involve BLAST on the server machine.

perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold> -b 3 -e 4

c) If the server machine has multiple processors available, then the user could run BLAST on multiple processors as follows:

perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold> -b 3 -e 4 -n <number of processors>

d) Copy the input, output, and working directories back to the desktop machine. Now, run steps 5,6, and 7 on the desktop machine.

perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold> -b 5 -e 7

e) Now, copy the input, output, and working directories back to the server machine and run step 8.

perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold> -b 8 -e 8

f) Now, copy the input, output, and working directories back to the desktop machine and run step 9.
perl start.pl -i <input_dir> -o <output_dir> -w <working_dir> -v 2 -l <log file> -c <config file> -a <algorithm> -t <threshold> -b 9

If the desktop and the server machines are on the same network such that the input, output, and the working directories can be accessed from both the machines, then the user does not have to copy these directories back and forth.

The above is just an example of how a user can run the pipeline in multiple stages.