BTEC 6304 Project

Genome Assembly Selection, Gene Prediction and Protein Annotation

Team Leader: Mengchuan Zhuang

Team Member:

Suman Nandy

Nam Nguyen

Yixi Wang

UNIVERSITY of HOUSTON

Github link:

https://github.com/ntnguyen63/fun-guy/tree/main

Software dependencies:

Canu, Flye, Busco v5.1.2, braker2, RepeatMasker, RepeatModeler, blast, GenemarkES, Prothint, Augustus

Most of these can be installed with conda but Genemark and Prothint may require manual installation. Also, RepeatMasker and RepeatModeler will need to be setup.

In short, make sure every individual software listed above can be run from the terminal.

Usage:

The program takes 4 inputs:

-s “sequencing data” – nanopore file

-gs “approximate genome size”

-l “species” – currently only support fungi input, bacteria and archaea may work but functionality is not tested

-db “protein database” – the protein database to annotate the genome from, in one fasta file

Python modules required:

Os, Sys, Argparse, Subprocess, Bio.SeqIO, Bio.SeqRecord, Bio.Blast.NCBIXML, Multiprocessing.

Funguy.py functionality:

This program takes input files and starts genome assembly with flye and canu, followed by comparison using busco to choose the optimal assembly for the downstream protein annotation.

Algorithm:

The **main** function takes the input variables (genome file, gene size and thread) as attributes and does the genome assembly. When genome assembly is done, the **run_busco** function will be used to activate the **run_busco_canu** function and **run_busco_flye** function to create the assembly quality reports. Then the **busco_result** function will select the optimal quality reports. In the end, the resulted genome data will be masked by the **repeatmask** function and the proteins will be annotated through the **run_braker** and the **annotate_proteins** function based on the masked genome. (if __name__ == "__main__": \ main () command was used to confirm the main function in this python module.)

Assembly.py

This file includes some functions needed in the **main** function mentioned above.

The **modules imported** are listed as below:

Shutil: offers some operations on files, will be used to remove or move files here.

Multiprocessing: a package that can spawn processes and allows the user to leverage multiple processors on a machine.

Description of each function:

Assemble_genome: This function takes the genome file and genome size variables and creates an output folder named "'input name'_outdir" in the same working directory (this function will check if there is a folder that has the same name, if it does, the function will ask if the user wants to delete the existed folder), then start the genome assembly by running **run_canu** function and **run_flye** function. If there is no exited folder with the same name, the function will run **run_canu** function and **run_flye** function directly.

run_flye: This function takes the corrected genome file (corrected by canu, will be explained later) and check if the 'fly_out' folder exists, if it does, remove it; else gets the working path and uses the maximum cpu number to run the flye and output the results to "fly_out folder". Then it will move the "fly_out" folder to the "'input name'_outdir" folder.

run_canu: this function will be run before the **run_flye** function since the **run_canu** function will correct the input genome file first. Once the genome sequence is corrected, canu will be run to assembly the genome sequence.

run_busco_canu: this function will get the path of the present working directory and check if the default output folder 'busco_out_canu' exists as mentioned above and run busco to create the quality report based on the result file of **run_canu** function then output the results to a new created 'busco_out_canu' folder.

run_busco_flye: this function will get the path of the recent working directory and check if the default output folder 'busco_out_flye' exists as mentioned above. After the checking step, it will run busco to create the quality report based on the result file of the **run_flye** function then output the results to a newly created 'busco_out_flye' folder.

run_busco: this function uses the multiprocessing module imported to run the **run_busco_canu** function and the **run_busco_flye** function together.

busco_result: this function will get the path to the result files made by the **run_busco_canu** function and the **run_busco_flye** function and extract the total complete score from the result files. Then the function will compare and select the one that has a higher complete score.

repeatmask: this function soft mask the repeated genome sequence to reduce the working data.

run_braker: this function will use prothint to predict and score hints of possible genes in the genome of interest by comparing the genome sequence to the reference protein database. Then braker will be used to predict and report the genes based on the output of prothint.

Annotate.py

This file includes the **annotation** function and its sub-functions needed in the **main** function mentioned above.

Annotate_proteins: use getAnnoFastaFromJoingenes.py to extract protein sequences from main_assembly.fasta.

Makeblastdb: create blast database with makeblastdb command from blast+.

Run_blastp: Run blastp command. The query is predicted proteins or other proteins and the database is the output database made by **Makeblastdb** function.

Seq_lookup_table: Extract name and sequence from fasta, ie predicted protein from braker

Go_through: Take the blast match and extract the protein description from title, separated by OS and a space " ".

Hits_from_blast_results: Attach the function into the protein name from the predicted protein list.

Annotate_proteins: Using all the functions above to annotate proteins from the assembly.