Wheat microRNA Pipeline Manual

Douglas Huang
Agriculture and Agri-Food Canada

April 16, 2015

Introduction

Welcome to the ECORC wheat microRNA pipeline, developed by Dr. Sylvie Cloutier and her research team! This pipeline can be used to effectively annotate conserved and novel microRNA found in wheat RNA samples.

The original pipeline was designed and created by Mahdi Rahman, a graduate computer science student at the University of Manitoba, and his supervisor, Dr. Michael Domaratzki. They can be reached at rahman.safiurbd@gmail.com and mdomarat@cs.umanitoba.ca respectively. The code that you are about to be using today is directly based on the original pipeline, but has been heavily modified, refactored, and documented by me, so if you have any questions or issues regarding the code or procedures, please direct them to me at dhdouglas.huang@gmail.com.

This manual is made for both programmers and non-programmers alike. If you are a beginner or haven't written a single line of code in your life, don't fret! Just follow the step-by-step instructions exactly. It does not matter if you do not fully understand the technical descriptions but nonetheless, please read the entire manual as it may contain important information between the lines! And for all you veteran coders, I will try to be as descriptive and articulate as possible, as I too know the pains of inheriting a legacy project.

Required Tools

- Linux (recommended)
- Bash Shell/Terminal
- GNU Parallel
- Biopython (Python library)
- BLAST+ standalone
- String::Approx (Perl module)
- edgeR (R package)
- bowtie2
- MAFFT
- FASTQC
- Geneious
- Git (optional)

Preparing the Workspace

Most of the required software should already be pre-installed on your computer. Otherwise, installation of any missing packages should be fairly straightforward for Linux distributions, using the “apt-get install” command mostly. The pipeline is compatible with Windows too, with the cygwin terminal, but all the code must be refactored, since it utilizes GNU Parallel, a UNIX-only tool (or you can use the old version of the pipeline without efficient parallel processing).

There should already be a local copy of the pipeline code on your machine, but if you wish to download the production copy or use another computer, you can clone the pipeline Github repository with the command:

```
git clone https://github.com/ECORC-Cloutier/
   mirna-pipeline.git
```

You must be authorized to push any commits to the repository.

Basic Operation

For most steps of the pipeline, there is a Python script that does the actual data processing, and a complementary shell script for automation and to input files. However, since GNU Parallel was implemented to optimize CPU performance, the bodies of these shell scripts have actually been written into functions, found in the “main.sh” file, and what remains in each separate shell script is simply a function call with parallel processing. Therefore, you will have to include the main.sh file in all of your working directories, as it contains the function declarations. If you are on Windows, the shell scripts will not work, since they use GNU Parallel, so you must create new shell scripts from the functions in main.sh or use the older set of code found in the repository that does not involve GNU Parallel.

In order to operate the pipeline, you need to have a basic knowledge of the Bash scripting language and know how to operate the terminal.

The three major aspects and commands you need to know are changing directories:

```
cd /path/to/directory #goes into a folder
cd .. #goes back to the previous directory
```

running a script:

```
bash script.sh #for shell scripts
python script.py #for Python scripts
```

and an understanding of wildcards (important!):

```
for file in *.fasta
do
  command
done
```

Wildcards, like "*" and "?" are place-holder names or letters than can be substituted in for anything, allowing shell scripts to iterate through a set of specific files in a directory. You may need to change them in each shell script. Comments in the code (indicated by #) have been left to guide you.

Knowledge of GNU Parallel is also important for those who want to make changes to the code:

```
source main.sh
export -f function
parallel function {} ::: *.txt
```

"source" imports functions from main.sh, "export" selects the specific function, and "parallel" is the actual call to GNU Parallel, which acts on the function and submits arguments that are delimited by ":::". In this example, GNU Parallel calls for the function to act on all files with .txt extension simultaneously.

Here is a quick tip that most beginners tend to forget: **You must be in the directory where the script is located in order to run it.** Change into the directory using terminal to run a script. Consequently, any files that you wish to process must also be in the same directory as the script. If you need any assistance, the Internet is full of wonderful reference guides and tutorials!

Procedure

WARNING: The scripts in this pipeline utilize wildcards for file names and involve a little bit of hardcoding as well. Never run a script without reading it beforehand to see if any modifications are necessary.

1 Pre-Processing Stage

1.1 Trim off adapter sequences

The raw files from the MiSeq contain an adapter sequence that needs to be trimmed off. The steps below utilize a script written by the BC Genome Science Centre and there is a readme.txt included about its trimming algorithm. First, we unzip the .fastq files, convert them to .qseq, run the trim, and finally convert them back. The output of this step are the files with ending "trimmed.fastq"

1. Create a new directory (folder) to store all the raw files.
2. Unzip the files into .fastq.
3. Copy the files main.sh, fastq2qseq_parallel.sh, fastq2qseq.pl, , qseq2fastq.pl, qseq2fastq_parallel.sh, trim_parallel.sh, adapter_trim.pl, and adapter_trim.pm into the directory.
4. Change into the directory using the terminal (this will be assumed in future steps) and run the script fastq2qseq_parallel.sh. Remember that this is a shell script:

```
cd /path/to/directory
bash fastq2qseq_parallel.sh
```

Several files with .txt extensions should now be produced.

5. Open the trim_parallel.sh and change the wildcard and adaptor sequence on line 7 if necessary. For the wildcard in this situation, you want something that includes the last three letters of the file name, as well as the extension. Then, run trim_parallel.sh.

6. Run qseq2fastq_parallel.sh.

NOTE: It is important to run these scripts sequentially.

1.2 Filter and convert .fastq files to .fasta format

Depending on the RNA extraction method, certain lengths of microRNA may need to be filtered out. Ask the lab technicians if this step is necessary for the data that you have been given.

1. If filtering is necessary, import files into Geneious and filter out all reads longer than "x" length. The adaptor trimming has automatically filtered out anything less than 14 base pairs already. Then, export the files as .fasta files.

2. If filtering is not necessary, either convert the files to .fasta using Geneious or copy convert_fasta_parallel.sh and convert_fasta.py into the directory and run convert_fasta_parallel.sh.

3. Open the files with FASTQC to assess data quality.

2 Initial Stage: Filtering

2.1 Extract unique sequences for each replicate

This step eliminates redundancies by grouping identical reads and adding up their read counts. Aggregate_whole.py takes the original .fasta file as

the input and produce a unique .fasta file, a summary file in .csv that records each unique sequence and calculates its normalized read count (reads per million) by $\frac{read\,count}{total\,reads} \times 1000000$, as well as an unaltered master.fasta.

1. Create a new directory storing the newly processed .fasta files. This shall be known as the working directory.
2. Copy aggregate_whole.py, aggregate_parallel.sh, and main.sh (main.sh is needed for every step later on) into the directory. Modify the number of days in aggregate_whole.py (replace "XDAYS").
3. Run aggregate_parallel. This will now create a new directory for each replicate.

2.2 Split .fasta file into 300 smaller files for processing

As the files are too large for the BLAST, this step divides each unique .fasta into 300 smaller segments. The number of splits can be changed in main.sh.

1. Copy split_parallel.sh and splitFasta.py into the working directory.
2. Run split_parallel.sh.

2.3 Prepare databases for BLAST

Here, we prepare to filter out unwanted information by BLAST by preparing the database files. The main database file needs to be divided into unique parts, and then those parts must be converted into a BLAST-able format. The database files used are updated frequently so it is necessary that you understand how to generate them (please refer to the "Code Maintenance" section at the end of the manual). The RNA BLAST file is a custom-created .fasta file from the Rfam database, so whenever a new version of the database is desired, you must email the curators of Rfam asking for a file with all RNA sequences from plants excluding microRNA". There is one additional database, for chloroplasts, which comes from NCBI.

1. Create a new directory (not within your working directory) and put the Rfam .fasta file inside, along with build_dbs.py and build_cat1.py.

2. Run build_dbs.py, with the file as an argument:

```
python build_dbs.py file.fa
```

   When the two output files are generated (with ".lncrna" and ".no_lncrna" extensions respectively), rename the file with .lncrna to "LNCRNA.fa".

3. Run build_cat1.py with the recently produced .no_lncrna file as the argument:

```
python build_cat1.py file.no_lncrna
```

   Rename the output file with extension .cat1 to "CAT1.fa". Rename the output file with extension .no_cat1 to "GENE.fa".

4. Go to the NCBI Nucleotide site, search for "triticum timopheevii chloroplast" and download the complete genome in .fasta format. Rename it to "CHLORO.fa".

5. Prepare the BLAST databases using the command:

```
makeblastdb -in file.fa -dbtype nucl
```

   where file.fa is the name of the database file (such as CAT1.fa), in terminal. After running this command, there should be three output files (.nin, .nsq, .nhr). Rename them to the parent name, but keeping the extension (for example, with CAT1.fa, rename outputs to CAT1.nin, CAT1.nsq, CAT1.nhr).

6. Repeat this process for each of the four databases (CAT1, GENE, LNCRNA, and CHLORO).

2.4 Filter out contaminating ncRNA sequences

In this step, we must remove anything that is not microRNA by BLASTing the dataset against all other types of RNA and the chloroplast genome. Four BLASTS will be conducted sequentially, thus the processing time is quite lengthy. For each of the 300 files, each BLAST will yield an .xml file, a .fasta file with extension "x", where x is a representation of the database name (like .c for CAT1), containing all sequences with positive hits, and a .fasta file with extension "nx", containing all sequences that passed filtering (like .nc for CAT1). Each BLAST takes the sequences that passed the previous database and BLASTs it again, hence first there is extension ".nc" (no CAT1), then ".ncg" (no CAT1 or GENE), ".ncgl" (no CAT1, GENE, or LNCRNA), and finally ".ncglch" (no CAT1, GENE, LNCRNA, and CHLORO).

1. Copy newFilter.py, ncrna_blast_parallel.sh, run_filter_cat.sh, run_filter_gene.sh, run_filter_lncrna.sh, run_filter_chloro.sh, and all the newly created database files (.nin, .nsq, .nhr) into the working directory.

2. Run ncrna_blast_parallel.sh.

2.5 Merge split files back together

After the BLAST, the divided files must all be recombined. However, we only care about the files that were BLASTed last (.ncglch extension) since they were BLASTed against each database and succeeded filtering. crush.py will take all 300 files created per BLAST and crush them together to create a "summary.smaller.csv" file.

1. Copy crush_parallel.sh and crush.py into the working directory.

2. Run crush_parallel.sh.

3. If you wish to also summarize the results from each of the BLASTs instead of only the last one, there are scripts available in the optional_steps folder. Copy over all the crush python scripts and simply run crush_other_parallel.sh.

2.6 Give consistent names to the sequences

It is important to annotate identical sequences with the same name across the entire dataset. For example, a sequence in day 7, replicate 2, heat stress, must have the same identifier as an identical sequence in day 10, replicate 1, control. In this step, pickle files, binarized files, are created for each day from the summary.smaller.csv file of each replicate (dumping all the sequences inside) and then compared against other pickle files for identical sequences. It will output a .csv file with extension ".cons", meaning that its naming has been made consistent.

1. Copy make_pickle.py into the working directory.
2. Open the script and change the array and for loop in the script to list however many days you have if necessary. You will see instructions inside the code.
3. Run make_pickle.py. Pickle files will be created for each day. Delete the ones for days that you do not have.
4. Create a directory for each day (i.e. day0, day1, etc.), within the working directory, and sort the replicates into their respective folders.
5. Copy consist_day0.py and rep_consist_day0.sh into the day 0 directory or whichever directory is numerically first.
6. If your first day is not day 0, open both scripts and make the necessary changes as indicated by the comments. Run rep_consist_day0.sh.
7. Open consist_day1_to_later.py and change the days array at the top if necessary. Then copy consist_day1_to_later.py and rep_consist_day1_to_later.sh into all the other day directories.
8. Run rep_consist_day1_to_later.sh for each day, starting from the lowest day number to the highest day number (order matters!), and inputting the current day number as a command-line argument:

```
bash rep_consist_day1_to_later.sh NUMBER
```

where NUMBER is the current day number (i.e. 3, 7 , 10, etc.). Remember that you must change into the directory first.

9. If the output .csv is too large and exceeds the limit for a spreadsheet browser (highly unlikely, only occurs with high-output NGS machines like HiSeq), it is possible to split the .csv file into parts of 500000. However, be warned that if you choose to do so, several wildcards will have to be modified further down (you'll have to figure it out on your own!). Create a directory with only the .cons files, copy csv_split.py and csv_split_parallel.sh from the optional_steps directory, as well as main.sh, and run csv_split_parallel.sh.

NOTE: File hierarchy is extremely important! Follow steps exactly or change the shell scripts.

2.7 BLAST sequences against miRBase

In this final step of the initial stage, mature and novel microRNA will be identified. The database file is from miRBase. First, make_master.py generates a .fasta file from the .cons .csv file for each replicate. Then, find_mature_0mm.py takes the master .fastas input to BLAST against miRBase, looking for 0 mismatch hits. It outputs several files, but most notably, "group" file that includes all the reads that match to a hit (used for multiple sequence alignment in the next stage) in both .fasta and .csv, a .fasta and .csv for all 0 mismatch conserved microRNA found (*_mature.fa), and a .fasta for all sequences that failed (*_1_4mm.fa). The failed .fasta file will be used as input to find_mature_1_4mm.py, which looks for 1-4 mismatch hits. It yields the exact same output as the previous script, a .fasta and .csv for 1-4 mismatch conserved microRNA (*_1_4mm_mature.fa) and a .fasta file for novel microRNA (*_1_4mm_novel.fa).

1. Prepare the BLAST database file by running the script mir_plantize.py and the "mature.fa" file from the FTP site of miRBase as an argument (recall 2.3, step 2), in a different directory.

2. Take the output, "MPC.fa", and create the BLAST database using the makeblastdb command (as seen in step 3).

3. Rename the newly created files "MPC" with their respective extensions (i.e. MPC.nhr).

4. Make a new directory within the working directory and copy all of the files from each replicate with ".cons" extension into it. If you are lazy, you can run this command in terminal while in the working directory:

```
find -name "*.cons" -exec cp {} /path/to/new
   /dir \;
```

5. Copy main.sh, master_parallel, make_master.py, find_mature_parallel.sh, find_mature_novel_parallel.sh, find_mature_0mm.py, find_mature_1_4mm.py, and all the database files with "MPC", including MPC.fa, into the new directory.

6. Run master_parallel.sh.

7. Run find_mature_parallel.sh. This will take some time.

8. Run find_mature_novel_parallel.sh. This will take equally as long.

2.8 BLAST results report generation

This step is technically not required for the continued operation of the pipeline, but produces the most important information; it generates .csv reports of successfully annotated microRNA and their respective reads per million and counts. This step takes the consistent summary .csv file for each replicate as input, matches the BLAST sequence to its reads per million count, and outputs a .csv file, for both 0 mismatch and 1-4 mismatch.

1. Create a new directory within your working directory and populate it with all the ".cons" files, "*_mature.csv", and "*_1_4mm_mature.csv" from the miRBase BLAST. You can use the find command from the previous step.

2. Copy main.sh, create_id_rpmlist.py and rpmlist_parallel.sh.

3. Run rpmlist_parallel.sh.

3 Secondary Stage: Identification

3.1 Align conserved sequences to the wheat genome

Now that all the BLAST has been performed, any potential conserved microRNA has been annotated. For extra validation, we now map these sequences onto the wheat genome using bowtie2. This step takes an extremely long time (over 48 hours for 12 MiSeq files). Although there is a function in main.sh (align_group()) that automates this process to run in parallel for all files, it is strongly recommended that you choose only the replicates that you wish to analyze and process them separately. You will first need to build the bowtie2 database. After, msa.py will take a group file as input and output .fasta and .bam for alignments and contigs.

1. Download the latest wheat genome as a .fasta/.fa file.
2. Create the alignment database with the command:

   ```
   bowtie2-build file.fa trit
   ```

   where file.fa is the name of your database file. Your new database files will be named “trit”. This will take a very long time.
3. Download the hairpin.fa file from the miRBase FTP site.
4. Create a new directory and move all of the aforementioned files into it, including main.sh, msa.py, and extract_precursor.py.
5. Copy all of the group_mature .fa files from the miRBase BLAST into the directory. You may use the find command.
6. In terminal, run:

   ```
   python msa.py file.fa database.fa
   ```

   where file.fa is a group file and database.fa is the genome file (full name).

3.2 Identify novel microRNA sequences

Currently, the identification of novel microRNA has yet to been completed. involves the use of RNAfold, a complex mathematical modelling tool for computational biology.

Nonetheless, at this point in time, it is safe to assume that all of the conserved microRNA have been successfully annotated.

4 Final Stage: Verification

DISCLAIMER: The following stage is still under development and has not been tested due to unavailability of data. Since a clear procedure has yet to be defined, no shell scripts have been made to automate the process and the current, but the current code has been made to be slightly modular.

4.1 Compare control replicates with stress replicates

Here marks the beginning of differential gene expression analysis. It involves a comparison of the microRNA abundance (as given by the reads per million) for a stress against a control. Thus, we use the files for 0 mismatches from the mirBase BLAST reports (2.8) as input and output a matrix.csv that compares all the control replicates and all the replicates of a stress for each day. A log file (*_sequence_log.txt) of all the species and their corresponding sequences is also created for steps further downstream.

1. Create a new directory for each day and within each day directory, create directories for each stress (not control).

2. Using any method of your liking, copy the BLAST reports for 0 mismatches (*rpm_blasted_0mm.csv) of the stresses into their respective directories. At the same time, copy the control replicates for each day into each stress directory for that day. For example, directory Day0/Heat/ will have all the heat replicates for day 0 and all the control replicates for day 0 inside.

3. From any location, run the script matrix.py:

```
python matrix.py /path/to/day/stress/
   NUM_REPS
```

   where /path/to/day/stress/ is the file path to a stress directory and NUM_REPS is the total number of replicates in the directory.

4. Repeat this for each stress for each day. Feel free to write a shell script to automate the process.

4.2 Perform differential expression analysis

In this step, we employ an R script to generate important metrics, as well as graphs and diagrams, on the previous comparisons made between controls and stresses. The matrix.csv files for each comparison are used as input and most significantly, a *_filtered.csv file is produced which contains only significant microRNA.

1. Run the script heat_dge.py on all the heat stress matrix files:

```
Rscript heat_dge.py /path/to/day/heat/
   heat_matrix.csv
```

   where /path/to/day/heat/ is the path to the directory containing the heat matrix file and heat_matrix.csv is the name of the heat matrix file.

2. Repeat with all the heat files for all the days.

3. Repeat process with light_dge.py for light files and uv_dge.py for UV files.

4.3 Find the microRNA target sequences

Using psRNATarget, a web database, we can find the target sequences of each microRNA species that was filtered by the differential expression analysis. First, a .fasta file is generated from the filtered.csv to upload onto the website. Then, the results, (in .txt format) are downloaded and parsed to output two final, resulting files which contain wheat targets and non-wheat targets respectively.

1. Run fasta
2. Upload
3. Run parse

The End

Congratulations! You've made it to the end. Give yourself a big pat on the back for all that work. Now, the onus is on you to maintain and optimize this pipeline. Best of luck!

Code Maintenance

The unfortunate truth is that nothing lasts forever. Sooner or later, parts of the pipeline, or even the entire thing, will have to be rewritten to account for updates, or modified for new procedures. There are certain areas that are more prone to changes, which you should be aware of:

The database files, used for both the BLAST and the multiple sequence alignment, are constantly changing. New sequences are added to these files almost daily, so newer versions are always being released. Unfortunately, it is not always as simple as just downloading the latest release since the pipeline code may have been hardcoded to not recognize newer types of data, or in a worst-case scenario, the text formatting and syntax within the files (such as the headers) may have been changed, rendering them incompatible with the pipeline. Therefore, it is always important to cross-reference new files with older ones and make changes to the code,

if necessary.

Here's a good example: for the creation of the MPC database for the miRBase BLAST, only certain database headers are given as parameters in the code (mir_plantize.py), meaning that the database is only populated by specific types of microRNA (e.g. only plants). When the database files are updated, there may be new species present that should be included into the BLAST database as well, thus necessitating a code change. Fortunately, this phenomenon is quite rare, so checking for only need to happen maybe once a year or so. For other database files, a quick glance over the document structure should suffice.

Other changes that my occur during the lifetime of the pipeline could be updates to the various software tools used, such as bowtie2 and MAFFT, though it is extremely unlikely that any functionality or method of use should change at all. As the pipeline is still young and in review, new modules may need to be added or procedures need to be redefined, meaning that the pipeline must be carefully restructured to ensure that the correct routing of input and output files.