miR-PREFeR: microRNA PREdiction From small RNAseq data

The miR-PREFeR pipeline is still under active development. To use the newest features from the pipeline, check the https://github.com/hangelwen/miR-PREFeR page and obtain the newest version. The current version is only tested under Python 2.6.7, Python 2.7.2 and Python 2.7.3 and should work under Python 2.6.* and Python 2.7.* (Tested platforms: Linux, Mac OS). It does NOT work under Python 3.0 currently. If you find any problem, please contact the author.

1. Required programs

To run the miR-PREFeR pipeline, the ViennaRNA package(tested under 1.8.5 or 2.1.2, 2.1.5) and samtools(0.1.15 or later. Tested under 0.1.18, 0.1.19) should be installed on the system. miR-PREFeR uses samtools commands to manipulate SAM and BAM alignment files, and uses RNALfold from the ViennaRNA package to do RNA secondary structure folding. The miR-PREFeR pipeline takes SAM alignment files as input, so an aligner such as Bowtie is also required (Not required by the pipeline, but needed for preparing the input data).

ViennaRNA package

The ViennaRNA package can be downloaded from http://www.tbi.univie.ac.at/~ronny/RNA/index.html.. The website provides precompiled packages for most popular Linux distributions. For these platforms, download the corresponding package and install it according to the platform package management system. If no precompiled package is provided for your platform, you can install it from source code. To install the package from source code, simply decompress the source code (e.g. type "tar xvf *.tar.gz) archive and cd into the folder, then:

```
./configure
make
make install
```

The above commands only work if the user has the root permission.

If the user does not have root permission. then the package can be installed to user specified locations (here I assume you install the program in directory /user/tools/ViennaRNA/):

```
./configure --prefix="/user/tools/ViennaRNA/" --without-perl
make
make install
```

After finish installing the package, **add the /user/tools/ViennaRNA/bin directory to the PATH environment variable.** Instructions on how to add a path to the PATH environment variable can be found at http://www.cyberciti.biz/faq/unix-linux-adding-path/ or just google "`add directory to PATH`".

NOTE: Because that RNALfold from the ViennaRNA package version 2.0.4 has a bug (If the input sequence has no valid folding structure, the program produces a segmentation fault), please make sure to use the newest version of the ViennaRNA package.

Samtools

Samtools can be downloaded from http://samtools.sourceforge.net/. Please follow the instructions from the package to install it. Please note the version 0.1.15 or later is needed (The pipeline uses the `samtools depth` command, which was introduced from version 0.1.15). One example set of commands to install samtools are given below. First, suppose you downloaded samtools-0.1.19.tar.bz2 and saved it in a folder with address /Users/xyz/. Then:

```
cd /Users/xyz/
tar jfx samtools-0.1.19.tar.bz2
cd samtools-0.1.19
make
```

If successful, you should see an executable program named "samtools" in the same folder. You need to add the samtools executable to the PATH environment variable.

For Mac OS users

Since it maybe difficult for some Mac OS users to successfully compile C/C++ programs, we provide several pre-compiled versions of samtools and RNALfold, which are put in the dependency/Mac/ folder. Mac OS users can first try to use the pre-compiled programs. If they do not work (This is possible), then try to compile from the source code by following the instructions in this section. To test whether a pre-compiled program works on your machine, change directory to the folder that contains the program, and then:

```
chmod +x RNALfold
./RNALfold -V
```

Here I used the pre-compiled RNALfold as an example. If you can see usage information from the output, then the program works on your machine. Then add the path of the RNALfold to the PATH environment variable, and RNALfold is correctly configured.

Note that in the dependency/Mac/ folder there are several folders named after specific Mac OS X version. If you can find the same version as your own system, then use pre-compiled programs under that folder. Otherwise, try to use programs which were compiled with lower Mac OS X version number.

2. Obtain and install the pipeline

The miR-PREFeR pipeline is hosted on Github (https://github.com/hangelwen/miR-PREFeR). To obtain the program, open an terminal window and execute the following command:

```
git clone https://github.com/hangelwen/miR-PREFeR.git
```

This makes an directory named 'miR-PREFeR' in the current directory, and clones the newest version of the miR-PREFeR pipeline code into the directory. The pipeline is ready to use.

If you do not have `git` on your system, simply go to https://github.com/hangelwen/miR-PREFeR/releases, download the zip file or the tar.gz file of the latest version.

The miR-PREFeR pipeline is still under active development, to use the newest features from the pipeline, check https://github.com/hangelwen/miR-PREFeR and obtain the newest version.

3. Test the pipeline.

The package contains a small dataset for testing whether the pipeline works after you download it. Please refer to HOW_TO_RUN_EXAMPLE.txt file to see more infomation about testing.

4. How to run the pipeline.

a. Prepare input data for the pipeline.

The miR-PREFeR pipeline needs the following input:

1. A fasta file, which contains the gnome sequences of the species under study.
2. one or more SAM files which contains the alignments of small RNAseq data with the gnome.
3. (Optional) An GFF (http://www.sanger.ac.uk/resources/software/gff/spec.html) file which lists regions in the gnome sequences that should be ignored from miRNA analysis.

a). Genome fasta file

Fasta format specification can be found at http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml. In miR-PREFeR, for the string following ">", only the first word that is delimited by any white space characters (whitespace, tab, etc) is used. For example, for the following sequence, 'ath-MIR773a' is used as the identifier of the seqeunce. **Thus, please ensure that all the sequences in the FASTA files have different identifiers.**

```
>ath-MIR773a MI0005103
AGGAGGCAAUAGCUUGAGCAAAUAAUUGAUUGCAGAAGUCCAUCGACUAAAGCUGUCACCUGUUUGCU
UCCAGCUUUUGUCUCCU
```

b). SAM alignment files

The miR-PREFeR pipeline takes SAM format alignment files. SAM alignment files can be generated by many aligners. Here we use Bowtie (http://bowtie-bio.sourceforge.net/index.shtml) as an example.

Step 1. Prepare RNAseq fasta files for Bowtie

a. Prepare RNAseq fasta files from uncollpased fasta files.

Because miR-PREFeR is able to use multiple small RNAseq data as input and utilizes information from multiple RNAseq samples to increase the performance of the miRNA loci prediction. **Thus, It's an requirement that the RNAseq reads data (fasta files that contain the RNAseq reads) be preprocessed by the provided `process-reads-fasta.py` script.** The script is under the `scripts` folder. The script takes a file that contains a list of sample/library names and an list of **uncollapsed fasta files** as input, and renames the reads in the input files, and produces a list of collapsed fasta files. Here `uncollpased fasta` means that identical reads in the fasta file have multiple entries.

For example, if you have 3 small RNAseq samples in uncollpased fasta format with names `SAMPLE1.fasta`, `SAMPLE2.fasta`, and `SAMPLE3.fasta`, and the samples are from 'root', 'flower', and 'salt', respectively, then first create a file with the following three lines:

```
root
flower
salt
```

If you name the file as `samplename.txt`, then run (assume you are in the directory where you installed miR-PREFeR. Otherwise, using the correct path to the process-reads-fasta.py script.):

```
python scripts/process-reads-fasta.py samplename.txt SAMPLE1.fasta SAMPLE2.fasta
SAMPLE3.fasta
```

Three new files with name `SAMPLE1.fasta.processed`, `SAMPLE2.fasta.processed`, and `SAMPLE3.fasta.processed` are produced **in the same folder as the input fasta files**. These three files are then aligned to the genome fasta file using Bowtie.

The following is a section of the processed fasta file from the root sample. The identifier of each read contains the sample name as the first part, then a sequential number rx as to identify the read, the last part is xN, where N is the depth (the number of occurrence) of the read. **Note that this is the required format of the identifiers of the reads. The read identifier in the SAM alignment file MUST follow the format. Otherwise, the pipeline stops at the very early stage and prints message complaining the format of the SAM files.**

```
>root_r0_x1
ACTACTGCAAGGGCTGGCTCAACCCGC
>root_r1_x5
TGGTTGCTGTCGCTGGTCGCTGGT
>root_r2_x1
CAAGGACAACAAGTGACGCCG
>root_r3_x186
ATAACCGTAGTAATTCTAG
>root_r4_x1
CCCATATTTTCTCTGAGCCTT
>root_r5_x1
ATACGGTTCGTTCTGG
>root_r6_x31
AACTGCGAATGGCTCATTAAAT
>root_r7_x1
```

```
AAAGGTCGACGCGGGCTCTGCCCG
>root_r8_x1
GATCCGGTGAAGTGTTCGGATC
>root_r9_x1
GGTAGAGATCGGAGG
>root_r10_x24
GTGGTTGTAGTATAGCGGTTAG
>root_r11_x2
TTTGAACGCAAGTTGCGCCCCAA
```

b. Prepare RNAseq fasta files from mirdeep/mirdeep2 format collapsed fasta files.

If you have fasta files in mirDeep, mirDeep2 format (Reads are already collapsed and the identifiers follow format "Identifer_xN", where 'Identifier' is an arbitrary string without any blank characters. 'N' is the number of occurrence (depth) of the read in the library.), we provide a script `convert-mirdeep2-fasta.py`, which is also under the `scripts` folder, to convert them to the required format for miR-PREFeR:

```
python scripts/convert-mirdeep2-fasta.py samplename.txt SAMPLE1.mirdeep.fasta
SAMPLE2.mirdeep.fasta SAMPLE3.mirdeep.fasta
```

Three new files with name `SAMPLE1.mirdeep.fasta.processed`, `SAMPLE2.mirdeep.fasta.processed`, and `SAMPLE3.mirdeep.fasta.processed` are produced **in the same folder as the input fasta files**. These three files are then aligned to the genome fasta file using Bowtie.

Step 2. Align the RNA-seq fasta files with Bowtie:

A script, `bowtie-align-reads.py`, is provided to align the read files in fasta format generated by `process-reads-fasta.py`, `convert-mirdeep2-fasta.py`, or `convert-readcount-file.py`. The script is under the `scripts` folder. Here is an example to align the processed fasta files generated in the previous stage using the script:

```
python bowtie-align-reads.py -f -r TAIR10.fas -t bowtie-index/ -p 8 SAMPLE1.fasta.processed
SAMPLE2.fasta.processed SAMPLE3.fasta.processed
```

The command first generates the bowtie index files in the bowtie-index/ folder for the TAIR10.fas reference sequences, and then uses 8 threads to align `SAMPLE1.fasta.processed`, `SAMPLE2.fasta.processed`, and `SAMPLE3.fasta.processed`. Unmapped alignments (bowtie outputs SAM files with both mappped and unmapped alignments.) are filtered using SAMtools (the -f opiton). The result files are in the same folder as the input files, with suffix ".sam". For more options available in `bowtie-align-reads.py`, please refer to the scripts/README-SCRIPTS.txt file, or run `bowtie-align-reads.py` with the `-h` option:

```
python bowtie-align-reads.py -h
```

For those who prefer to use bowtie directly (for example, users want to more bowtie options), please refer to Section Conducting alignment using Bowtie. The section contains the steps to do alignments using bowtie and the format of the SAM files required by the miR-PREFeR pipeline.

c). The (optional) genome annotation file in gff3 format

For a genome with known annotations, some regions in the genome can be excluded when doing the miRNA analysis. For example, many species have protein coding sequence (CDS) annotations, so there is no need to run miR-PREFeR on those regions. Most plant genomes contain highly repetitive regions such as transposons, these regions should also be excluded. These regions should be listed in a GFF file (http://www.sanger.ac.uk/resources/software/gff/spec.html). Note that for the gff file, you should not contain introns, because miRNAs could be in introns.

b. Prepare a configuration file for the pipeline.

The miR_PREFeR.py script takes a configuration file as input. The configuration file lists all the information (such as the location of the SAM files, genome fasta file, etc) needed to run the pipeline. An example configuration file with detailed descriptions of each option is in the example data. The options are also explained in the following section.

1. PIPELINE_PATH: The path of the miR-PREFeR pipeline.
2. FASTA_FILE: The path of the genome sequences in fasta format. Absolute path preferred.
3. ALIGNMENT_FILE: The path of the RNAseq alignment files in SAM format. Absolute path preferred.
4. GFF_FILE: The path of the optional GFF file. Absolute path preferred.
5. PRECURSOR_LEN: The max length of a miRNA precursor. The default is 300
6. READS_DEPTH_CUTOFF: The first step of the pipeline is to identify candidate regions of the miRNA loci. If READS_DEPTH_CUTOFF = N, then genomic position that the mapped depth is smaller than N is not considered. Default value is 20.
7. NUM_OF_CORE: Number of CPUS/Cores avalible for this computation. If commented out or leave blank, 1 is used.
8. OUTFOLDER: Outputfolder. If not specified, use the current working directory.
9. NAME_PREFIX: Prefix for naming the output files. For portability, please DO NOT contain any spaces and special characters. The prefered includes 'A-Z', 'a-z', '0-9', and underscore '_'.
10. MAX_GAP: Maximum gap length between two contigs to form a candidate region. Default value is 100.
11. CHECKPOINT_SIZE: The pipeline makes a checkpoint after each major step. In addition, because the folding stage is the most time consuming stage, it makes a checkpiont for each folding process after folding every CHECKPOINT_SIZE sequences. If the pipeline is killed for some reason in the middle of folding, it can be restarted using `recover` command from where it was stopped. The default value is 3000.

c. Run the pipeline.

With the configuration file ready, the pipeline can be used to predict miRNAs. The miR-PREFeR pipeline can be run as following:

```
python miR_PREFeR.py [options] command configfile
```

Currently, the following options are available:

1. **-h**: Show help information.
2. **-L**: Generate a log file. From the log file you can find the status during the running of the pipeline. **Recommend to always use this option.**

3. **-k**: After finish the whole pipeline, do not remove the temporary folder that contains the intermediate files. This will save disk space. If it's not specified, you can delete the temporary folder after getting the result.

`command` could be one of `check`, `pipeline`, `prepare`, `candidate`, `fold`, `predict`, and `recover`. (Run `python miR_PREFeR.py -h` to see help on options and commands.)

1. **check:** Check the presence of the depended programs (RNALfold and samtools), and check the recovery information (Shows which stage the previous computation on the same data was ceased. See the `recover` command in 6.).
2. **pipeline:** Run the whole pipeline. That is, run `prepare`, `candidate`,`fold` and `predict` sequentially. This is the normal way to run miR-PREFeR.
3. **prepare:** Run the first step of the pipeline. This step prepares some data files needed in the following steps.
4. **candidate:** Identify possible candidate regions. This step can ONLY be run if the `prepare` step has been finished on the configfile file.
5. **fold:** Fold the candidate regions. This step can ONLY be run if the `prepare` and `candidate` steps have been finished on the configfile file.
6. **predict:** Predict miRNA loci. This step can ONLY be run if the `prepare`, `candidate` and `fold` steps have been finished on the configfile file.
7. **recover:** Tries its best to recover an unfinished job and continues to run from the unfinished step it was ceased. This is designed to easily continue a job other than re-run the whole pipeline from start. For example, if a job was kill half way (This happens sometime. For example, the job takes too long time and the user killed it. Or, if the job was run on an cluster and was killed halfway because some resources exceed the max values.) In these cases, one does not need to run the pipeline from start, the job can be continued by using the `recover` command:`python miR_PREFeR.py -L recover configfile`

NOTE on job recovery: To recover and continue a job, the intermediate output files in the temporary folder (See the Output section) should not be removed. All files needed to do recovery are in this folder.

4. Output

The miR-PREFeR pipeline produces several files which contains information about the predicted miRNAs:

1. An HTML file that contains a table which summarizes the results of the miRNA predictions by miR-PREFeR. It contains the number of predictions, the length distribution of the predicted mature miRNA sequences, the list of all pre-miRNA sequences and their secondary structures, and the detailed read mapping profiles against the pre-miRNA sequences. In addition, this table provides search links to mirBase. An example output HTML page can be found at http://goo.gl/YQSCP8.
2. A csv file that contains detailed information about each prediction. The information in the table is the same as in the HTML file. This is provided for easier downstream processing/analysis.
3. A list of read mapping result files in the folder `readmapping`. Each file corresponds to an miRNA and shows the reads mapped to the precursor region.
4. A gff3 format annotation file that contains the predicted miRNAs.
5. Two fasta files that contain the mature sequences and the precursor sequences of the predicted miRNAs.

6. A `.ss` file that contains the sequence and the predicted structure of each miRNA.
7. A file named `miRNA.stat.txt` that shows the basic statistics of the mature sequences.

An example output folder that contains all the files can be found at http://goo.gl/f4Jb2p

During the running of the pipeline, there is a folder named NAME_PREFIX + "_tmp" in the output folder. The folder contains intermediate files generated by the pipeline. If the job is finished without interruption, the folder is deleted by default (Specify the `-k` option can keep the folder, but this is usually not needed.). **If the job is interrupted half way, then the folder is not removed by default, so that one can run the job in the `recover` mode to continue the computation. (Because all files needed to do recovery are in this temp folder, so if one wants to recover an interrupted job, please do not remove the folder. The folder will be removed automatically once the whole pipeline is successfully run.).**

5. Conducting alignment using Bowtie.

a1. indexing the genome sequences:

```
bowtie-build -q -f TAIR10.fas bowtie-index/TAIR10
```

a2. conduct the alignment. Here are example bowtie commands for the same files processed in the previous step:

```
bowtie -a -v 0 -p 8 -m 30 -S bowtie-index/TAIR10  -f SAMPLE1.fasta.processed > SAMPLE1.sam 2>
SMAPLE1.log
bowtie -a -v 0 -p 8 -m 30 -S bowtie-index/TAIR10  -f SAMPLE2.fasta.processed > SAMPLE2.sam 2>
SMAPLE2.log
bowtie -a -v 0 -p 8 -m 30 -S bowtie-index/TAIR10  -f SAMPLE3.fasta.processed > SAMPLE3.sam 2>
SMAPLE3.log
```

The -v 0 option allows no mismatch in the alignment. The -m 10 option discards reads that can be mapped to more than 30 positions. The -p 8 option uses 8 threads to do the alignment. More options of the parameters can be found at http://bowtie-bio.sourceforge.net/index.shtml.

Requirement for the SAM file format:

The pipeline requires the input SAM alignment files:

a. The SAM files **MUST have headers**. This is a example header:

```
@HD     VN:1.0  SO:unsorted
@SQ     SN:Chr1 LN:30427671
@SQ     SN:Chr2 LN:19698289
@SQ     SN:Chr3 LN:23459830
@SQ     SN:Chr4 LN:18585056
@SQ     SN:Chr5 LN:26975502
@PG     ID:Bowtie       VN:0.12.8       CL:"bowtie -a -v 1 -p 10 -S bowtie-index/TAIR10 -f
../SAMPLE1.fasta.processed"
```

You can use "`samtools view -H yoursamfile`" to check whether your SAM file has the header section. Bowtie produces headers for SAM format alignment files by default.

b. The flag column (the second column of the alignment section) of the SAM file **MUST be in a integer.**. The flag generated by Bowtie/or generated by using "`samtools view`" command, by default, is an integer. Samtools provides options to output SAMfiles with Hex (-x option) or string (-X option) format flags. If your SAM file uses these two types of flags, you either need to generate your SAM file again, or convert it to the required format.

c. The read identifier (the first column of the alignment section) must be in the required format as described in the fasta file preparation section. The `process-reads-fasta.py` and `convert-mirdeep2-fasta.py` scripts can help you generate fasta files of the right format.

The miR-PREFeR pipeline checks the SAM format and if the format is not correct, it prints messages about which format problems it encountered and stops analysis.