Fluidigm2PURC Documentation

Release 0.1.0

Paul Blischak

Feb 05, 2019

Contents

Fluidigm2PURC is a set of Python scripts for processing paired-end Illumina data generated from sequencing double-barcoded PCR amplicons.

CHAPTER 1

Tutorials

1.1 Quick Start

This **Quick Start Tutorial** will walk you through every step of downloading, installing, and running the Fluidigm2PURC pipeline. The details of each step can be found in the main documentation.

Requirements:

- Python (we suggest using Miniconda)
- Python modules: pandas, numpy, biopython, cython
- C, C++ compilers (Linux should be good, Mac OSX needs Xcode and the **Command Line Tools**)
- zlib (needed to compile Sickle; may already be present)
- PURC (available on Bitbucket)

Note: We have tested our scripts on Python 2.7, 3.5, and 3.6. However, **PURC has only been tested with Python 2.7.** We have also worked with others researchers who had trouble getting things run with Python 3. Therefore, we recommend using Python 2.7.

1.1.1 1. Downloading and Installation

Python

The code below will walk you through downloading and installing a Python distribution using Miniconda, as well as all of the Python packages that needed to use Fluidigm2PURC.

```
# Get Miniconda for your operating system (Mac or Linux)
# Answer yes to the questions the Installer asks
# These commands will download Python 2.7 for Mac OSX
```

(continues on next page)

(continued from previous page)

```
curl -O https://repo.continuum.io/miniconda/Miniconda2-latest-MacOSX-x86_64.sh
bash Miniconda2-latest-MacOSX-x86_64.sh

# Install packages with conda or pip command
conda install numpy pandas biopython cython
# pip install numpy pandas biopython cython
```

PURC

PURC is available on Bitbucket and can be cloned and installed using the code below.

```
git clone https://bitbucket.org/crothfels/purc.git
cd purc && ./install_dependencies.sh

# while in the PURC directory, add it to your PATH
# It's best to add the PATH to your .bash_profile
export PATH=$(pwd):$PATH
```

If you are on a Linux computer, you may have to run the `install_dependencies_linux.sh` script instead. The Bitbucket repository for PURC has more details about installation as well.

We have also included a modified version of the *purc_recluster.py* script as part of our pipeline (*purc_recluster2.py*). The only difference is that it conducts fewer iterations of the chimera detection and clustering steps. If you would like to use it, make sure that move or copy it from the Fluidigm2PURC folder into the main PURC folder.

Note: For the PURC scripts to work, they need to be present in the main PURC folder that was cloned from Bitbucket. The reason for this is that the scripts reference all its dependencies using file paths that are relative to the main PURC folder. These scripts also need to be made available in your bash `PATH` variable (see code above).

Fluidigm2PURC

Fluidigm2PURC is available on GitHub and can be cloned and installed using the code below.

```
git clone https://github.com/pblischak/fluidigm2purc.git
cd fluidigm
make && sudo make install
```

The haplotyping script, *crunch_clusters*, can optionally call the programs Mafft and Phyutility. If you would like to use these tools, make sure that you install them on them your machine and add them to your PATH.

1.1.2 2. Running *fluidigm2purc*

The *fluidigm2purc* script will process a set of paired-end FASTQ files that have been demultiplexed using the program dbcAmplicons and will output a single FASTA file for each locus present using sequence header information in the format required by PURC. As an example, let's say that we have our paired-end data in the files `FluidigmData_R1.fastq.gz` and `FluidigmData_R2.fastq.gz`. To run these data through the script, all we would need to run is:

```
fluidigm2purc -f FluidigmData
```

This will filter/trim the reads using the program Sickle, merge the paired-ends (if possible) using FLASH2, and then write everything to a FASTA file in a new directory named `output-FASTA/`. If we want to tweak some of the settings for the parameters that are used to filter/merge reads, we can specify them using command line flags (type `fluidigm2purc -h` to see options). In addition to the FASTA files, the fluidigm2purc script outputs two other files: (1) a table containing all individuals where their ploidy level can be specified (`output-taxon-table.txt`) and (2) a table with per locus error rates (`output-locus-err.txt`).

1.1.3 3. Running PURC

If we `cd` into the `output-FASTA` directory, we can run PURC using its *purc_recluster.py* script to do sequence clustering and PCR chimera detection. If you want to use the *purc_recluster2.py* script, make sure you move or copy it into the main PURC folder. Also, because *purc_recluster2.py* only does three iterations of chimera detection and clustering, it only requires that two clustering thresholds be specified using the `-c` argument (rather than the usual four).

The code below will loop through all of the FASTA files in the `output-FASTA` directory and will write all of the output to a new directory named `output-PURC/`.

```
cd output-FASTA

for f in *.fasta
do
  purc_recluster.py -f $f -o output-PURC \
                    -c 0.975 0.99 0.995 0.997 -s 2 5 --clean
done
```

1.1.4 4. Processing PURC clusters

The script to infer haplotypes from the clusters returned by PURC is called *crunch_cluster*. If you `cd` into the directory where we wrote all of the PURC output, you can loop through each locus and analyze each one in turn. If you know the ploidy levels for your organism, you can add them to the `output-taxon-table.txt` file.

The code below will use the locus names in the `output-locus-err.txt` file to loop through all of the output files from PURC to infer haplotypes. It will also realign the sequences clustering Mafft (`--realign`), clean the sequences using Phyutility (`--clean 0.4`), and will only return unique haplotypes for each sample.

```
cd output-PURC

for l in $(tail +2 ../../output-locus-err.txt | awk '{print $1}')
do
  crunch_clusters -i ${l}_clustered_reconsensus.afa -s ../../output-taxon-table.txt \
                  -e ../../output-locus-err.txt -l $l --realign --clean 0.4 --unique_
↪haps
done
```

1.1.5 5. Downstream

Once all of the loci have been haplotyped, some of them may still contain an excessive amount of gaps from being aligned to bad clusters (or because reads never merged). We can use Phyutility to clean these up one more time.

Example:

```
# Remove sites with more than 40% gaps
phyutility -clean 0.4 loc1_crunched_clusters.fasta
```

CHAPTER 2

Main Documentation

2.1 Getting Started

2.1.1 Requirements

- Python
- Python modules:
  - numpy
  - pandas
  - biopython
  - cython
- C, C++ compilers (Mac users need Xcode Command Line Tools)
- zlib (needed for Sickle)
- PURC

2.1.2 Installation

```
git clone https://github.com/pblischak/fluidigm2puc.git
cd fluidigm2purc
make
sudo make install
```

The Makefile will clone Sickle and FLASH2 from GitHub and will compile them from source into a folder called `deps/`. You'll need to have C and C++ compilers to do this. Typing `sudo make install` will copy the fluidigm2purc scripts and all of the dependencies to `/usr/local/bin` so that you can run everything from anywhere on your computer.

2.2 Running *fluidigm2purc*

The *fluidigm2purc* script combines the tasks of read filtering/trimming based on quality scores, merging filtered paired-end reads, and conversion of the resulting output to the proper format for running through PURC. Run `fluidigm2purc -h` to see options.

Steps:

1. Read trimming with Sickle
2. Paired read mergind with FLASH2
3. Conversion from FASTQ to PURC-compatible FASTA format ("PURCifying")

Each of the steps listed above can be run individually as well. That way, if you want to rerun one of the step with different setting, you don't have to start from scratch. A particular step can be specified using the `-p` flag and the name of the step you want to run (`sickle`, `flash2`, `PURCify`). By default, the script will run all three steps (`all`). The only mandatory option is the prefix for the paired-end FASTQ files (e.g., 'FluidigmData' for the files FluidigmData_R1.fastq.gz and FluidigmData_R2.fastq.gz), which is given using the `-f` flag.

```
# By default, the script will run all three steps (i.e., --program all)
fluidigm2purc -f FluidigmData

# To only run Sickle
fluidigm2purc -f FluidigmData -p sickle

# To only run FLASH2
fluidigm2purc -f FluidigmData -p flash2

# To only run the PURCifying step
fluidigm2purc -f FluidigmData -p PURCify
```

The final output is a directory named `output-FASTA` that has a single FASTA file for each locus that was present in FASTQ files used as input. The `output` part of the directory cab be substituted with whatever is supplied by the `-o` argument (default=output).

2.2.1 PURCifying

The conversion from FASTQ to FASTA is straightforward because FASTA only uses the first two lines of every four line sequence entry in the FASTQ file. The important bit here is that we grab the relevant information from the sequence header in the FASTQ file and print it so that it is compatible with PURC. The things that we want are the taxon name and the locus name. These are added by dbcAmplicons when the Fluidigm data are demultiplexed. **Taxon names and locus names can't have spaces in them.** The code splits on spaces first, then on colons (":") so that it can grab the taxon and locus names (this is specific to the way the Fluidigm data are processed by dbcAmplicons). Merged reads from FLASH2 are processed first. Unmerged reads are then read in together and are artificially combined with eight N's in between ("NNNNNNNN").

2.2.2 Additional outputs

`output.log`

The *fluidigm2purc* script will output a log file that lists the taxa and loci found during the processing of the FASTQ files. It also lists the command line arguments that were used to generate the analysis.

output-taxon-table.txt

A two-column table listing each taxon and its ploidy level is generated so that users can specify what the ploidy level of the sample is when processing the clusters output by PURC (see the section on *cluster crunching*). By default, all of the ploidy values are set to `None`.

Example:

```
Taxon    Ploidy
taxon1   None
taxon2   None
.
.
.
taxonN   None
```

output-locus-err.txt

fluidigm2purc will also calculate the per locus error rate using the PHRED quality scores in the FASTQ files. It does this by calculating the average error for each read mapping to a locus, followed by the overall average across reads. These error values are used in the *cluster crunching step* to determine if a cluster output by PURC is sequencing error or not.

Example:

```
Locus    Error
loc1     0.002341729
loc2     0.032134829
.
.
.
locN     0.000967257
```

2.3 Running PURC

After running the *fluidigm2purc* script, you will have a new directory named `output-FASTA`. The next in the process is to run PURC. We will do this for each FASTA file individually using a Bash `for` loop. The script from PURC that we will run is called *purc_recluster.py*. Make sure that it is in the main PURC folder and also that it is in your `PATH`. The code below is an example of what this would look like if you are running things on a Unix-flavored computer.

```
cd output-FASTA/

# Loop through all of the fasta files and run purc_recluster.py
for f in *.fasta
do
    purc_recluster.py -f $f -o output-PURC \
                      -c 0.975 0.99 0.995 0.997 -s 2 5 --clean
done
```

We have also included a modified version of the *purc_recluster.py* script as part of our pipeline (*purc_recluster2.py*). The only difference is that it conducts fewer iterations of the chimera detection and clustering steps. If you would like to use it, make sure that move or copy it from the Fluidigm2PURC folder into the main PURC folder. Also, it also only requires two clustering thresholds for the `-c` option.

2.4 Processing PURC Clusters

The *crunch_clusters* script takes the output from PURC and will determine the haplotype configurations for your samples using the sizes of the resulting clusters. These clusters are the output from *PURC*. To see all of the options for running the script, type `crunch_clusters -h`.

During this part of the pipeline, we use the taxon-ploidy table to determine the number of haplotypes that should be output (e.g., a diploid should have 2, a tetraploid 4, etc.). We also use the per locus error rates table to calculate the probability that a given cluster is a sequencing error. The *Haplotyping Tutorial* provides details on the mathematical model that we use for this step.

```
cd output-PURC/

crunch_clusters --input_fasta loc1_clustered_reconsensus.afa \
                --species_table ../../output-taxon-table.txt \
                --error_rates ../../output-locus-err.txt --locus_name loc1
```

We can also treat the locus as haploid by specifying the `--haploid` flag. This can be used for chloroplast or mitochondrial loci, as well as for nuclear loci when all we want is the primary cluster.

To go through all of the clustered loci, we can use a bash script and a for loop to analyze each locus. If the loci under consideration are haploid, add the `--haploid` flag.

```
# List all of the loci using the error rates file
for l in $(tail +2 ../../output-locus-err.txt | awk '{print $1}')
do
  crunch_clusters -i ${l}_clustered_reconsensus.afa -s ../../output-taxon-table.txt \
                  -e ../../output-locus-err.txt -l $l
done
```

Some other useful options during this step include realigning the sequences using Mafft (just add `--realign` to your command). We can remove gappy sites using Phyutility as well. This can be done by adding the `--clean <%>` flag. Just substitute the percent of gaps allowed per site that you want to use for cleaning. We also also have an option to only return unique haplotypes using the `--unique_haps` flag.

Note: We have run into some issues with species' names when using Phyutility. First, it doesn't like the semicolons that PURC uses to delimit the different parts of the sequence identifier (species names, cluster #, cluster size). We use `sed` to substitute underscores for the semicolons automatically. Other characters such as dashes have created issues as well because they get automatically substituted for underscores and no longer match the original names. If you are running into issues with not getting any output from the `crunch_clusters` script, make sure you check the names of the species in all of the files it writes to make sure that things are being inadvertently changed.

CHAPTER 3

Extras

3.1 Determining Haplotypes

3.1.1 Unknown ploidy-level

Inferring haplotype configurations for individuals with unknown ploidy levels involves identifying which clusters are likely to be "real" haplotypes, and which ones are likely to be errors. We do this by considering a set of models that range from treating all clusters as errors, to one where all clusters are real haplotypes. The models in between successively treat the next cluster in the ordered set as a real haplotype (sorted by size). For an individual with N clusters, there are $N + 1$ models to test. Each of these models has H real haplotypes $(0, \ldots, H)$ and $N - H$ errors $(H + 1, ..., N)$. The likelihood for each of these models is the sum of the clusters sizes $(C1, ..., CN)$ times the log probility that they are sequencing errors (ϵ) or not $(1 - \epsilon)$. The likelihood for a model with H real haplotypes is given by:

$$\ell_H = \sum_{i=0}^{H} C_i \times \log(1 - \epsilon) + \sum_{j>H}^{N} C_j \times \log(\epsilon).$$

To determine the most likely haplotype configuration, we calculate how much the likelihood increases over the previous model when another haplotype is added (the likelihood is monotonically increasing). We also normalize these differences by the total change in likelihood from the model with $H = 0$ to the model with $H = N$. If this value is less than a given cutoff (we use a default of 0.10), the previous model is treated as the best configuration. Since the cluster sized are ordered, the increase in the log-likelihood will always be smaller for any additional haplotypes.

3.1.2 Known ploidy-level

To infer the maximum likelihood haplotype configuration using integer partitions, we use a multinomial likelihood that uses the size of each cluster being considered as a haplotype. For an individual with ploidy level K, we take the first K clusters sorted by size and calculate the likelihood for a given partition as follows: each entry in a partition, P, contains the number of times that a particular haplotype is represented in the configuration. Given cluster sizes C_1

through C_K and a sequencing error rate of ϵ, the log likelihood for a partition P is:

$$\ell_P = \sum_{i=0}^{|P|} C_i \times \log\left(\frac{P[i]}{K}\right) + \sum_{j>|P|}^{K} C_j \times \log(\epsilon).$$

Here $|P|$ represents the size of the partition.

3.2 Fluidigm2PURC on Docker

We have made the Fluidigm2PURC pipeline available as a Docker image in the hopes that it will facilitate its use by providing all dependencies pre-installed. This also allows any researcher with the Docker software installed on their computer to use our pipeline (e.g., Fluidigm2PURC won't work on Windows without using Docker). Details on Docker itself can be found on the main website: link.

To obtain the Fluidigm2PURC image, first download Docker for your computer (if you haven't already done so). Then, launch a terminal window and use the following commands to get the software:

```
docker pull pblischak/fluidigm2purc
```

To run an analysis with Fluidigm2PURC, launch a Docker container that is running the Fluidigm2PURC image:

```
docker run -it pblischak/fluidigm2purc
```

The above line of code will get you running inside a Docker container with everything that you need to run Fluidigm2PURC. However, you will also need to link your local files to the container. This can be done using the `-v` option. To start, navigate to the folder with your paired-end reads (R1 and R2) that you want to analyze with Fluidigm2PURC. Then, use this command to link that folder to the Docker container running Fluidigm2PURC:

```
docker run -it -v $(pwd):/home pblischak/fluidigm2purc
```

If you type `ls`, you should see your files available for analyzing. All analyses that you run in the container will also write those files to the directory from which you launched the container. This way, everything that you do will automatically be available on your computer outside of Docker.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search