**Overview of analysis**

This protocol is written for users who wish to analyze sequence data with the *dbcAmplicons* analysis pipeline. Raw sequence files from the MiSeq will need to be configured by the miseq\_pipeline.py script to fit the expectations of *dbcAmplicons* before the analysis can begin. The sample sheet from the run is reconfigured to support the output of four individual reads (Read 1, Index 1, Index 2, Read 2) and CASAVA (the Illumina-specific post-run processing software) is then run using the reconfigured sample sheet. The output from this step is four individual reads ready for analysis using *dbcAmplicons*.

**Configuring raw MiSeq data for input into the pipeline**

The following items are necessary for the proper execution of the pipeline:

* Python version 2.X (this protocol written for 2.7.5)
* A properly formatted sample sheet named SampleSheet.csv and the RunInfo.xml sheet
* Raw data generated from sequencing, before CASAVA is run
* Access and read permissions to the raw data
* Installed and executable CASAVA (this protocol written for 1.8)

Navigate to the desired directory and execute the miseq\_pipeline.py script, giving the first parameter as the raw data from the run and the second parameter as the name of the desired output directory. The sample sheet from the run must be named SampleSheet.csv and it must be in the run output folder, along with the RunInfo.xml sheet.

An example command (written in the shell terminal) run in the desired directory is as follows:

**python miseq\_pipeline.py 150325\_M01380\_0037\_000000000-AEMV5 Output\_directory**

where 'python' is the call to execute the script, '150325\_M01380\_0037\_000000000-AEMV5' is the name of the directory that holds the MiSeq run information, and 'Output\_directory' is the prefix for the desired output. The script will generate and run CASAVA within the output directory until completion. Once the process is complete, the reads are ready for analysis.

**Analyzing dual-barcoded amplicon sequence data using *dbcAmplicons***

There are 4 levels of analysis within the *dbcAmplicons* package. The package was developed for command line usage to analyze high volumes of microbial 16S, LSU, and ITS sequences. All components of the package are designed to flow directly into the next with no intermediate formatting steps required. For a detailed vignette using sample data provided with the *dbcAmplicons* package, please refer to the github page within the doc folder: <https://github.com/msettles/dbcAmplicons.git>.

*Installation*

Minimum requirements for the *dbcAmplicons* package: Python 2.X is required to run any and all of the

elements in the package. Some applications within the package rely on **flash2** or the **RDP classifier** to complete, so the user must download and install the external tools.

All executable scripts are located in the github repository <https://github.com/msettles/dbcAmplicons.git>

The **flash2** executable and installation instructions can be found at <https://github.com/dstreett/FLASH2.git>. The **RDP** **classifier** may be found at <http://sourceforge.net/projects/rdp-classifier/>. The user must install the classifier.jar and its dependencies in the RDPtools package.

*Mac OSX*

Mac users should first download and install **Xcode** and then download the app via a terminal by:

git clone https://github.com/msettles/dbcAmplicons.git

cd dbcAmplicons

sudo python setup.py install

dbcAmplicons

If installation was successful, this will bring up the usage arguments.

*Linux*

For Linux users, some python dependencies may be required before the installation is successful. Root privileges are also required to install these dependencies. To install, open a terminal and type:

sudo apt-get install python-dev

And type the sudo password at the prompt. Then, download the app via a terminal by:

git clone https://github.com/msettles/dbcAmplicons.git

cd dbcAmplicons

sudo python setup.py install

dbcAmplicons

If installation was successful, this will bring up the usage arguments.

***dbcAmplicons******preprocess***

*dbcAmplicons* *preprocess* takes four fastq files (Read 1, Index 1, Read 2, and Index 2), the assigned barcode indices for each sample, the sequences of the template-specific primers, and the individual sample names along with an assigned project folder. *dbcAmplicons preprocess* will then assign each read in the fastq file to a specific project according to both barcode and primer sequences, and return 2 fastq files with all of the identifying information in the read header. If a project directory is specified, *dbcAmplicons preprocess* will create a directory with the indicated output prefix.

At a minimum, *dbcAmplicons* *preprocess* requires a properly formatted reference barcode sequence file to demultiplex raw sequences. For maximum separation and identification of the reads, the following three inputs should be used, all in a tab-separated text format:

1. Reference barcode sequences file.

2. Reference primer sequences file.

3. Sample metadata file with sample IDs, primer IDs, and barcode IDs.

Examples of each of these sheets may be found in Appendix A. *dbcAmplicons preprocess* will automatically run a validation of the inputs before any reads are separated. The barcode and primer sheets will be checked against the sample metadata file to make sure that any assigned barcodes and primers exist within the reference files.

The syntax for usage is as follows:

dbcAmplicons preprocess [-h] -B barcodesFile [-d BARCODEDIFF]

[-P primerFile] [-D PRIMERDIFF] [-e PRIMEREND]

[-S sampleFile] [-q MINQ] [-l MINL]

[-b BATCHSIZE] [-O PREFIX] [-U] [-u] [-v]

[-1 read1 [read1 ...]] [-2 read2 [read2 ...]]

[-3 read3 [read3 ...]][-4 read4 [read4 ...]] [--test]

[--keepPrimers] [--debug]

The user will give the paths to each of the input files. Default settings for primer and barcode distances may be changed, but it is not recommended to set them to be less stringent unless the user has good reason to do so. The final outputs of *preprocess* include a text file listing the barcodes that were identified and two fastq files for each project that was specified.

***dbcAmplicons join***

The next step in analysis is to overlap the reads to create longer sequences. *dbcAmplicons join* is the step that requires the **flash2** executable in the user path for overlapping. The process will check for the executable and if it is not found the user will be prompted to download and install. When the reads are overlapped *dbcAmplicons join* will automatically discard reads that have extremely low quality scores, which indicates the presence of primer dimers that are not biologically relevant. Usage is as follows:

dbcAmplicons join [-h] [-O PREFIX] [-u] [-x NUM] [-t NTHREADS]

[-v] [-1 read1] [-2 read2]

The minimum required inputs are the paths to the preprocessed read 1 and read 2 that the user wants to overlap. Default settings are recommended. There may be cases where the overlapping criteria are too stringent, especially if there are large amounts of errors on the ends that prevent overlaps. The user must determine which settings would be appropriate in that case.

The outputs from the *join* process include 3 fastq files, one of the overlapped sequences (named extendedFrags.fastq) and two of the unpaired sequences (named notCombined\_1/2). The process will also generate histograms showing the size and frequency of the overlapped sequences. If the user finds large quantities of unexpected sequence lengths there may be contamination issues within the sample, and classifying the reads will likely give an erroneous result. However, if the amplicons are within the expected size range the user can proceed to the next step in the pipeline.

***dbcAmplicons classify***

Once the reads have been overlapped, they can be classified by 16S, ITS, or fungal LSU sequence using the RDP classifier. The *classify* process will check for the RDP classifier similarly to how the *join* process will check for the flash2 executable. Organisms are classified to their closest taxonomic rank that meets a 50% bootstrap match. Usage is as follows:

dbcAmplicons classify [-h] [-b BATCHSIZE] [-q MINQ] [-l MINL]

[-p PROCS] --rdpPath PATH [-g <arg>] [-t TRAIN]

[-O outputPrefix] [-1 read1 [read1 ...]]

[-2 read2 [read2 ...]] [-U single [single ...]][-v]

[--debug]

The output from the *classify* process is a single fixrank file that can then be processed with *dbcAmplicons abundance.*

***dbcAmplicons abundance***

Within the *abundance* process, the fixrank file is reformatted to produce three outputs that parse the classified data according to user-specified criteria. Usage is as follows:

dbcAmplicons abundance [-h] [-r <arg>] [-t VALUE] [-m VALUE]

[-M VALUE][-S FILE] [-O FILE\_PREFIX] -F FILE [FILE ...][-b] [-v] [--debug]

The user will determine the lowest classification levels that are determined, the largest and smallest amplicon sizes to be considered within the classification, and the threshold bootstrap value for a classification cutoff. These values are dependent on the quality and diversity of the samples as well as the user's own criteria. The minimum input would be the fixrank file if the user does not want to change any of the default settings.

The outputs include the following three text files. The abundance.txt file lists the taxon abun-

dance with the read counts. The proportions.txt lists the proportions of reads per sample, and the

taxa counts.txt lists a summary of the lowest level classification and the number of counts for that level.

**Comparison of *dbcAmplicons* and QIIME**

QIIME is one of the most commonly used applications for analyzing double-barcoded amplicons (cite). However, QIIME's limitations include poor documentation for user analysis and a reliance on clustering OTUs to classify amplicons, which may not accurately reconstruct true biological variation (cite). Additionally, QIIME does not have the flexibility offered by *dbcAmplicons* to analyze amplicons of different sizes within the same sequencing run, which is a limitation when the researcher is targeting multiple regions such as ITS, LSU, and 16S for a community analysis. QIIME also does not support multiple possible primer sets for analysis on the same sample which occurs when the researcher is attempting to introduce “spacer” base pairs to increase diversity within the run.