Introduction

This is the manual for the clustering pipeline, in this manual the steps and options to successfully cluster with the pipeline are explained. For the pipeline and the manual two datasets are available for testing, the Katwijk and Maasland ITS1 fungal datasets (provided by Barbara Gravendeel). Each dataset contains a reduced representation of larger datasets in order to keep clustering and identification times low for the manual, you can use these datasets to run the pipeline and compare the output with the manual.

In order to use the pipeline some basic knowledge is needed about Linux and the command line (terminal), since the pipeline is exclusively ran from commands. Online tutorials such as: http://code.google.com/edu/tools101/linux/basics.html, should provide information on the linux commands in order to use the pipeline.

The output produced by the pipeline (fasta sequences for each cluster, the blast results and cluster results) are saved in either in plain text files or comma separated value files. These files can be opened in normal text editors (notepad, wordpad, gedit) and the comma separated value files can be opened with spreadsheet programs such as excel, openoffice calc and R.

Installation and acquiring software

Python and python scripts

In order to run the pipeline two things are necessary; The pipeline python scripts and additional software that is needed by the pipeline (the actual cluster programs).

The python pipeline scripts can be obtained from the following link: https://github.com/Y-Lammers/Cluster-pipeline or github link https://github.com/Y-Lammers/Cluster-pipeline.git.

In order to run these files a recent version of python (2.7 or 3.2) should be installed on your system (most linux distributions come standard with python). You can check your version of python by starting python from the terminal with the command:

```
python
```

If you have a version of python installed that is lower than 2.7 or 3.2 you can install a new version of python from either your package manger of choice (such as apt-get for ubuntu based systems) or by downloading the source code from http://python.org/download/.

The pipeline uses some features from biopython in order to handle fasta files, alignments and genbank data. Biopython can be obtained from either the package manager or from http://biopython.org/wiki/Download.

Additional software

In order to use the pipeline several third party programs are needed, these are:

Cluster programs

At the moment four different cluster programs are supported by the pipeline, these are: octupus, usearch (Edgar, 2010), TGICL (Pertea *et al.,* 2003) and cd-hit (Weizhong and Adam, 2006). These programs can be acquired from the following sites: http://www.drive5.com/usearch/nonprofit_form.html (usearch, a free license can be obtained from the author of the program), http://weizhong-lab.ucsd.edu/cd-hit/download.php (cd-hit), http://octupus.sourceforge.net/ (octupus) and http://compbio.dfci.harvard.edu/tgi/software/ (TGICL).

NCBI blast+
NCBI blast+ (Camacho *et al.*, 2009) is used to identify clusters with a local blast search. The program can be obtained from either the package manager (usually its listed as 'ncbi-blast+') or from: ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/.

Setting up the pipeline

When the pipeline is ran for the first time it will automatically search for the above mentioned programs on your computer. In some cases one of the programs cannot automatically be found (for example due to a failed installation or placement in an obscure folder). You can manually add the paths to these missing programs in the paths.txt file in your main pipeline folder.

Basic usage of pipeline

Default cluster of data + identification

The pipeline in its most basic form requires two commands; a sequence file in fasta format with the sequences that need to be clustered and an output directory in which all the result file will be stored.

The pipeline command will look like:

python pipeline.py --input_file fasta_file --outdir_dir results_directory

This command will take the input sequences from the fasta file, cluster them at 97% sequence similarity with the program uclust (if uclust is installed, otherwise an error will be returned) and identify the clusters that contain at least ten sequences by blasting them against the NCBI nucleotide database.

A simple cluster run with the Katwijk ITS 1 dataset will have the following command:

python pipeline.py --input_file Katwijk_ITS1.fasta --outdir_dir Katwijk_simple

This command will produce the following files:

File name	**Description**
Katwijk_ITS1_otus.txt	File generated by the pipeline, contains the cluster results in the QIIME format.
cluster_stats.csv	Contains a overview of the cluster results such as the number of clusters, singletons and reads.
clust_rep.fasta	A fasta file with the representative sequences for the clusters.
blast_result.csv	The blast results for each representative sequence.

The cluster_stats.csv output for the Katwijk_ITS1.fasta cluster run looks like:

Katwijk_ITS1_otus.txt	
Number of sequences	1000
Number of clusters	124
Cluster time	00:00:00
Size of cluster	number of clusters for size
1	51
2	27
3	12
4	3
5	6
6	1
7	1
8	4
9	1
10	1
11	3
12	1
13	3
17	1
18	1
27	2
30	1
36	1
44	1
78	1
129	1
263	1

The cluster_stats.csv file provides a quick overview on the number of clusters, the size and distribution of these clusters. This information can be useful when dealing with larger datasets or when trying to optimize the cluster settings for a certain dataset.

The Katwijk_ITS1_otus.txt (or any other input_sequence_otus.txt file) contains information about the clusters themselves. It provides a list of clusters (number 1 to many) and the raw reads that make up this cluster. For example

```
5	bb1bb_GRJJ1HH03CX6D8length324xy109
	bb1bb_GRJJ1HH03DEHHBlength324xy127	bb1bb_GRJJ1HH03DLCFVlength320xy135
6	bb1bb_GRJJ1HH03DBYP5length329xy124
7	bb1bb_GRJJ1HH03C3SGBlength326xy115	bb1bb_GRJJ1HH03DIL4Hlength324xy132
```

Cluster 5 contains 3 reads, cluster 6 is a singleton with 1 read and cluster 7 contains 2 reads.

The clust_rep.fasta file contains all the representative fasta sequences (see section 'Selecting clusters and picking representative sequences' for more details). Such a representative sequence looks like;

```
>bb1bb_GRJJ1HH03DCDWMlength249xy125_cluster_#:82_length_cluster:13_
AAGAATCGCCCCGTTTTGAAATGGGTTCTATTCCCAAACCGTGTATACATACCTTTGTTG
CTTTGGCAGGCCGCCTCTTAGGCGTCGGCTCCGGCTGACTGCGCTTGCCAGAGGACCCAA
ACTCTTTGTTTAGTGATGTCTGAGTACTATATAATAGTTA
```

The fasta header contains information about the cluster: the '_cluster_#:82' section describes the cluster number (in this case 82), this number can be used to find the cluster in the Katwijk_ITS1_otus.txt file, this way the input sequences of the cluster can be traced. The '_length_cluster:13_' section of the header describes the size of the cluster, this cluster is based on 13 input sequences.

The blast.csv file finally provides the blast results for each representative sequence. These blast results include the blast hit, accession code (when dealing with online blasting), match length, e-value, bit-score and the percentage match among others.

Timing the cluster / identification run

With linux it is possible to time how long it takes to run a certain command with the time option. By placing time in front of a command such as the cluster run you'll get information on the duration of the cluster run after it is done. The basic cluster command with the time option will look like:

```
time python pipeline.py --input_file  fasta_file --outdir_dir results_directory
```

Multiple input fasta files

The --input_file can accept multiple fasta input files, by default these multiple input files will be combined into one sequence set and clustered together. Each input file in the command needs to be separated from each other by a space, for example a default cluster run with multiple input files will look like:

```
python pipeline.py --input_file fasta_file_1 fasta_file_2 fasta_file_3
--out_dir multiple_results_directory
```

For example, the Katwijk and Maasland ITS sets can be clustered together with the following command:

```
python pipeline.py --input_file Katwijk_ITS1.fasta Maasland_ITS1.fasta
--out_dir 2_ITS1_sets
```

The cluster_stats.csv file shows that the cluster ran has been carried out with twice the number of sequences (2000 rather than a 1000) and produced more than twice as many clusters (259 rather than 124).

Cluster settings and program

By default the data is clustered with the program uclust at 97% similarity, these settings can be changed if needed.

In order to change the cluster program specify the --program parameter (valid options are: octupus (default), usearch(*), cdhit and tgicl). Use the following command to cluster with usearch:

```
python pipeline.py --input_file fasta_file1 --output_dir usearch_run
--program usearch
```

* The latest supported version for usearch is version 6.0.307. the older versions of usearch (5 or earlier) use a different set of input parameters, these versions are still usable by the pipeline but the --program parameter should be set to 'usearch_old'.

The similarity parameter changes at which similarity sequences are placed in the same cluster, higher similarities make the clusters more specific, lower similarities makes the clusters more general. At default the data is clustered at 97% similarity which is commonly used to cluster data at species level. To change the similarity use the --similarity parameter. For example:

```
python pipeline.py --input_file fasta_file --output_dir 90_run --similarity 0.90
```

This command clusters the fasta_file to clusters of 90% similarity.

The usearch and 90% similarity settings have the following results on the Katwijk ITS dataset:

Default settings		**uSearch as cluster program**		**Clustered at 90% similarity**	
Number of sequences	1000	Number of sequences	1000	Number of sequences	1000
Number of clusters	124	Number of clusters	127	Number of clusters	92
Cluster time	00:00:00	Cluster time	00:00:00	Cluster time	00:00:00
Size of cluster	number of clusters for size	Size of cluster	number of clusters for size	Size of cluster	number of clusters for size
1	51	1	52	1	34
2	27	2	27	2	20
3	12	3	14	3	9
4	3	4	2	4	1
5	6	5	5	5	2
6	1	7	3	6	2
7	1	8	3	7	1
8	4	9	3	8	3
9	1	10	1	9	1
10	1	11	3	10	2
11	3	12	1	11	4
12	1	13	2	12	1
13	3	15	1	13	2
17	1	17	1	18	1
18	1	20	1	27	1
27	2	25	1	31	1
30	1	27	1	35	1
36	1	30	1	36	1
44	1	36	1	41	1
78	1	37	1	48	1
129	1	88	1	102	1
263	1	108	1	130	1
		263	1	263	1

Test run

Prior to identifying your clusters (which can take a long time when dealing with a lot of clusters) it can be helpful to do a test cluster run. The test run will generate the cluster_stats.csv file which can be used to check the number and distribution of the clusters.

To start a test cluster run set the --pipeline parameter to 'test', for example:

```
python pipeline.py --input_file fasta_file --outdir_dir test_run
--pipeline test
```

This command will cluster the data with uclust (default) at 97% similarity (default). The test results are saved in the cluster_result.txt file, this file can be found in the output directory (in this case the test_run folder).

The cluster_results.txt file will be generated for every cluster run, even if the --pipeline parameter isn't set to 'test', however setting the parameter to 'test' forces the program to stop after clustering and skip the identification steps to save time.

The test parameter can be used in combination with --similarity and --program parameters to test the cluster results for custom settings, for example:

```
python pipeline.py --input_file fasta_file --outdir_dir custom_test
--pipeline test --similarity 0.90 --program cdhit
```

This command will give the test cluster results for a cluster run at 90% similarity with the program cd-hit.

Selecting clusters and picking representative sequences

Depending on your dataset and the used setting the cluster run can produce a lot of data, in order to identify these clusters filtering can take place (to remove singletons for example) and a representative sequence needs to be obtained for each cluster.

The clusters can be filtered based on the minimum number of reads present in a cluster. A minimum size of two reads per cluster allows you to filter out the singletons (only one read per cluster) from the dataset. By default all clusters with less than 10 reads are removed, this setting can be changed with the --min_size parameter. The following command for example performs a default cluster run but saves all clusters (including singletons):

```
python pipeline.py --input_file fasta_file --output_dir all_cluster
--min_size 1
```

The results from a test run (see previous chapter) can help to set the --min_size parameter to a value that filters out the ”unreliable” small clusters.

For cluster identification a representative sequence is needed for each clusters, this sequence will be used for blasting (either online or local). By default a consensus sequence is drawn from the cluster results to act as a representative sequence, however this option is not available for all programs (tgicl and cdhit). A second option is to pick a random sequence from each cluster to act as a representative sequence, this method is supported by all programs but at lower similarity clustering it might not represent the entire cluster.

The cluster method can be picked with the --pick_rep setting. For example to use the random picking method use:

```
python pipeline.py --input_file fasta_file --output_dir consensus_rep
--pick_rep random
```

Identification

When a representative sequence is obtained it can be identified by either blasting it against a local database or the NCBI genbank database, both methods have their pros and cons, see table 3.

Method	**Pro’s**	**Cons**
NCBI genbank nucleotide blast	Can use the vast amount of sequence present at genbank for identification. The NCBI Taxon database will be used to expand the blast hit with additional classification data for the blast hit.	Blasting against the NCBI genbank database consumes a lot of time, identification of a 1000+ sequences can take up several hours.
Local blast	Quick compared to online blasting, custom database can be used that might be more relevant for the project goal.	Local blasting won’t benefit from the taxonomic information online blasting provides.

By default the pipeline will identify the clusters by blasting them online against the NCBI genbank nucleotide database. The blast method can be changed by the --blast command. Local blasting requires the user to specify a local reference database with the --reference command. For a local blast run a command can look like:

```
python pipeline.py --input_file fasta_file --output_dir local_blast --blast local
--reference reference_sequences
```

The reference database can be either a fasta file or a genbank file. Fasta files can cause the NCBI Blast+ program to crash if there are vertical bars ('|') present in the fasta header, if these are present in the fasta file either manually remove them in a text editor or use the --accession command. For example:

```
python pipeline.py --input_file fasta_file --output_dir local_blast_accession --blast
local --reference reference_accession_set --accession yes
```

The blast results from either the local or genbank blast are stored in the blast.csv results file, this wile will be placed in the output directory.

Filtering

The blast results from the blast.csv can be automatically filtered based on the blast percentage match and the blast match length with the --blast_percentage and the --blast_length settings. The --blast_percenage setting accepts a minimum percentage for a blast hit to be considered valid, the --blast_length is for a minimum blast length. Both commands can be used separately from each other, for example to only filter by percentage a command will look like:

```
python pipeline.py --input_file fasta_file --output_dir 97_blast
--blast_percentage 97
```

Or if the set will be filtered by both percentage and length:

```
python pipeline.py --input_file fasta_file --output_dir 97_150_blast
--blast_percentage 97 --blast_length 150
```

The original blast file will not be edited, instead the filtered blast results will be saved in the file 'blast_filtered.csv' which will also be placed in the output directory.

Comparing input files

Multiple input files

The pipeline can be used to directly compare the cluster results from multiple input files, this can prove useful when comparing multiple samples / sampling sites or other types of data that one might want to compare.

The final output of this type of analysis will be a list of clusters formed by the analysis, with added blast results (both local and online blast possible) and the number of reads from each sample file that contributed to said cluster.

Tagging

Normally when dealing with multiple input files the pipeline will simply merge them into one file and cluster them together. For the comparison analysis its necessary that the pipeline can distinguish the reads from the different input files. To do this the reads from the different files need to be tagged. The pipeline will automatically add a unique tag to the fasta headers of the input files, which the pipeline will use later on in the analysis.

The --pipeline setting can be set to ‘cluster’ to automatically tag the input files and set up the pipeline for a cluster / identify / comparison run, the command will look like:

```
python pipeline.py --input_file fasta_file1 fasta_file2 --output_dir comparison
--pipeline cluster
```

The comparison results will be saved in the cluster_input_seqs.csv file in the output directory. The last two columns of the file will indicate how many reads each input file contributed to the cluster.

A run with the ITS set will look like:

```
python pipeline.py --input_file Katwijk_ITS1.fasta Maasland_ITS1.fasta
--output_dir comparison --pipeline cluster
```

A subset from the output will look like:

Cluster	Katwijk_ITS1.fasta	Maasland_ITS1.fasta
TXN9SV_bb3bb_GZTTMNM01AJ51Jlength=320_cluster_#:11_length_cluster:49_	0	49
ET7YHF_bb1bb_GRJJ1HH03C0M9Wlength323xy112_cluster_#:22_length_cluster:19_	19	0
TXN9SV_bb3bb_GZTTMNM01AGXG1length=241_cluster_#:112_length_cluster:64_	45	19

ET7YHF_bb1bb_GRJJ1HH03C736Ilength329xy120_cluster_#:24_length_cluster:11_	11	0
TXN9SV_bb3bb_GZTTMNM01AF6WXlength=295_cluster_#:63_length_cluster:18_	0	18

The cluster setting can be used in combination with the normal pipeline settings such as cluster similarity and filtering options, a more elaborate run can look like:

```
python pipeline.py --input_file fasta_file1 fasta_file2 --outdir_dir full_run --pipeline
cluster --program usearch --similarity 0.95 --min_size 2
--blast local --reference ref_file --blast_percentage 97
```

Additional settings

Read data filtering

Prior to clustering the raw-read data can be filtered with the pipeline. Filtering is based on either the minimum length of the input reads, or the pipeline can remove duplicate reads.

For filtering based on size, use the --min_length and --max_length settings, the length settings accept an integer and removes all sequence reads shorter or longer than the setting. For example to remove reads smaller than 200 or longer than 500bp use:

```
python pipeline.py --input_file fasta_file --outdir_dir no_small --min_length 200
--max_length 500
```

It is not necessary to provide both a min and max length value, to only filter out the sequence shorter than 200 bp it is enough to provide only the --min_length parameter.

Duplicate sequences can be removed from the input set with the --duplicate setting. By default this setting is set to ‘no’, by changing it to ‘yes’ all duplicate sequences will be removed (first copy is preserved). For example:

```
python pipeline.py --input_file fasta_file --output_dir no_dup
--duplicate yes
```

TGICL multicore

The cluster program TGICL can utilize multiple cores. This can be specified with the --cores command. For example to use 4 cores for the analysis use:

```
python pipeline.py --input_file fasta_file –output_dir multicore --program tgicl
--cores 4
```

Full list of options:

These are the full options that are available for the pipeline, these can also be obtained from the terminal with the following command:

```
python pipeline.py -h
```

The output will look like:

```
usage: pipeline.py [-h] [--input_file fasta files [fasta files ...]]
                   [--output_dir output directory] [--pipeline pipeline]
                   [--filter filter sequences] [--length minimum length]
                   [--duplicate remove duplicates] [--program cluster program]
                   [--similarity sequene similarity for clustering]
                   [--blast blast method]
                   [--reference reference files [reference files ...]]
                   [--accession genbank accession]
                   [--blast_percentage minimum blast percentage]
                   [--blast_length minimum blast length]
                   [--pick_rep otu sequence picking]
                   [--min_size minimum OTU size]
                   [--cores # cpu cores for tgicl]

Pipeline to process 454 reads, clusters and identifies

optional arguments:
  -h, --help            show this help message and exit
  --input_file fasta files [fasta files ...]
                        Enter the 454 sequence fasta file(s)
  --output_dir output directory
                        Enter the output directory (full path)
  --pipeline pipeline   The way the 454 reads will be processed (only relevant
                        if multiple input files are used): merge (input files
                        are merged in a single file) / cluster (input files
                        are tagged and clustered together, bassed on tags the
                        origin of reads in clusters can be traced) / test (run
                        the cluster step and write the cluster info to a
                        output file without identifications) (default: merge)
  --filter filter sequences
                        Filter the sequences based on length or duplicates
                        (yes / no) default: no
```

--length minimum length
Filter out sequences smaller then the minimum length
--duplicate remove duplicates
Remove duplicate sequences from the dataset (yes / no) default: no
--program cluster program
The cluster program that will be used to cluster the 454 reads: uclust / cdhit / usearch / usearch_old / tgicl / octupus (default: octupus)
--similarity sequene similarity for clustering
Sequence similarity threshold used for clustering (default: 0.97)
--blast blast method The blast method used for identifying the reads: genbank / Local (default: genbank
--reference reference files [reference files ...]
Reference file(s) used for the local blast search
--accession genbank accession
Does the reference file contain accession codes in the fasta header (indicated by the |'s) yes / no (default" no)
--blast_percentage minimum blast percentage
Filter out the blast hits under the minimum blast percentage
--blast_length minimum blast length
Filter out the blast hits under the minimum blast length
--pick_rep otu sequence picking
Method how the OTU representative sequence will be picked: random / consensus (default: consensus) *note* consensus mode is not supported by cd-hit and tgicl clustering, these settings will automatically select the random mode
--min_size minimum OTU size
minimum size for an OTU to be analyzed (default: 10)
--cores # cpu cores for tgicl
number of processors used for the tgicl cluster analysis (default: 1)