Manual to the biodiversity sample clustering and identification pipeline:

Introduction

This is a manual for the biodiversity sample clustering and identification pipeline that I built. 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) and the UNITE fungal reference database (Abarenkov *et al.,* 2010). 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 run 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 as either 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.

The pipeline

Introduction

The cluster pipeline combines several python scripts with a few dedicated software packages for clustering and identifying sequences. The goal of the pipeline is that the user only needs to run the pipeline with one command and a set of parameters to get the cluster results, rather than manually running the separate programs and going through the troubles of converting the necessary files into the different file formats. A flowchart of the different python scripts can be found in figure 1.

Pipeline.py

The pipeline.py script is the main script of the pipeline, this script accepts the user commands and uses them to run either the other pipeline scripts or a series of bash commands. Though the clustering and raw data processing is dealt with by other scripts, the pipeline script is used to set up the correct folders and file handling.

Paths.py

The pipeline uses several programs for either clustering of sequences, identifying them or multiple sequence alignments. The path.py script uses the linux search command to find the system paths to these programs (if they are installed) and saves them in a text file. The several subscripts of the pipeline use the paths text file to locate and run these programs.

Filter.py

The user input sequences can be filtered with the filter.py script, this script can remove sequences with a length under a certain threshold or remove duplicate sequences from the dataset. The filtered set of sequences is saved and used by the other downstream scripts in the pipeline.

Tag_fasta_files.py

When the user submits multiple input sequence files these will be merged into a single file before clustering. This means that based on the fasta headers it can be difficult to track from which file a sequence came. If the different input files correspond to different samples, sampling sites etc. or contains other information that might be lost upon merging, the tag_fasta_files.py script can be used to add a unique tag to each sequence based on the input file. Based on the tags the source of the sequences in the merged file can still be traced back to the original input files. This can be useful for comparing different samples in a single cluster run.

Cluster.py

The cluster programs such as uclust, usearch and cd-hit are run with the cluster.py script. The cluster script takes the user settings and (merged) input files and uses them to set up the correct commands for the desired cluster program and run it.

Cluster_to_txt.py

Each cluster programs has its own type of output, in order to analyze the cluster information the output first has to be normalized to simple pars-able output. The cluster_to_txt.py script takes the output from the cluster programs and converts it into the QIIME output format.

Pick_otu_rep.py

In order to identify each cluster a fasta sequence is needed, based on the input sequences and the QIIME cluster file these cluster sequences can be obtained. The pick_otu_rep.py script can provide a representative sequence for each cluster based on three different methods: A random sequence from the cluster is used to represent the entire cluster, a consensus sequence represents the cluster, or a consensus sequence based on a subset of sequences from the cluster.

Each sequence picking method has its pros and cons.

- Random sequence picking is by far the fastest option for all sizes of clusters, however at low identity clustering it might not provide the most representative sequence for each cluster.
- The consensus sequence provides the most representative sequence for each cluster, however calculating the consensus sequences requires a multiple sequence alignment (done with muscle), this can (depending on the sequence length and number of sequences in each cluster) take up a vast amount of time and computer resources.
- The combined method works similar to the consensus method, however rather than making a consensus sequences based on all sequences in the cluster it randomly picks a number of sequences (user defined), aligns the subset of sequences and creates a consensus sequence based on the subset. The combined method provides a more accurate representative sequence for each cluster than random picking but consumes less time and resources then a full consensus sequence.

The programs used in the pipeline are capable of producing there own consensus sequences for the clusters, this is a lot faster then the method currently present in the pipeline. If a consensus sequence for the clusters is vital and your dealing with a large dataset, running these cluster programs separately from the pipeline might be helpful. Incorporating the way the cluster programs produce there own consensus sequences might be implemented in the future.

Blast.py

The blast.py script identifies the clusters by blasting the representative cluster sequences against the NCBI Genbank nucleotide database. Several sequences are blasted against the Genbank database in parallel to keep identification time low. For each blast process the best blast hit is saved together with the blast match information (match percentage, match length, e-value, bit-score etc), for each hit the genus and species names is retrieved as well together with the full biological classification (if possible) by connecting to the NCBI Taxonomy database.

Custom_blast_db.py
Identification based on a local reference set is carried out with the custom_blast_db.py script. This script uses the NCBI Blast+ software to set up a local blast database based on a reference file and blasts the cluster representative sequences against this database. Local blasting is faster than the Genbank blast since it requires no connection to other servers and removes the need for up- or downloading information, but unlike the Genbank blast it will not provide additional taxonomic information (except for the unite fungal reference dataset, this dataset contains all the relevant taxonomic information.).

Filter_blast.py
The blast results from either the local or Genbank blast can be filtered with the filter_blast.py script. The filter script can filter out low quality blast hits based on the percentage match or match length information. The user can set these parameters to a desired value and all blast hits that do not meet the criteria are replaced by a general 'no identification' hit.

Cluster_freq.py
The cluster_freq.py script is used to compare the clusters from the different input files (if there are multiple input files and the user wants to compare these). Based on the tags added to the sequences by the tag_fasta_file.py script, and the cluster file the number of sequences that make up each clusters and from which input file they come from is calculated. The results are combined with the blast identification to provide an overview that contains both the identification of the clusters and how many reads each sample contributes to said cluster.

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 three different cluster programs are supported by the pipeline, these are: uclust, usearch (Edgar, 2010) 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) and http://weizhong-lab.ucsd.edu/cd-hit/download.php (cd-hit). uclust is no longer supported by the author and has been merged into the usearch project, however the original uclust code might still be available on other sites.

Muscle
The pipeline can (depending on the settings) use the multiple sequence alignment program Muscle (Edgar, 2004). The program muscle can be installed from either the package manager or can be obtained from: http://www.drive5.com/muscle/downloads.htm.

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 run 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.
he pipeline command will look like:

```
python pipeline.py --input_file  fasta_file --out_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 --out_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 an 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 will 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 --out_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: uclust (default), usearch or cdhit). Use the following command to cluster with usearch:

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

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 --out_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 --out_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 --out_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 --out_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 random sequence is drawn from each cluster to act as a representative sequence, but a consensus can also be generated for each cluster. See table 2. for details on the different picking methods.

Method	Pros	Cons
random	Fastest picking method.	Representative sequence might not resemble the entire clusters at low similarity cluster runs (less than 90% for example)
consensus	Most accurate representative sequence for a cluster.	Depending on the cluster size and number of clusters this can take a long time to compute. This setting should be avoided if there are clusters present that contain more than 500 sequences or the total number of clusters exceeds a 1000.
combined	A combination of random picking and a full consensus. An x number (specified with the `--rand_cons` setting) of sequences is picked from a contig and a consensus is created from this subset. This gives a more accurate representation than a random representative sequence and is faster to compute than a full consensus.	Though more accurate than random picking the quality of a representative sequence still declines at lower similarities.

The cluster method can be picked with the `--pick_rep` setting. For a consensus representative sequence use:

```
python pipeline.py --input_file fasta_file --out_dir consensus_rep
--pick_rep consensus
```

The combined setting requires an additional setting to specify the number of sequences a consensus will be created from, the `--rand_cons` setting. By default this setting is set to 10, a higher number of sequences will make the consensus more reliable but slower to compute, a lower number of sequences will return a less accurate consensus but will be faster. For example:

```
python pipeline.py --input_file fasta_file --out_dir combined_20
--pick_rep combined --rand_cons 20
```

This will create a combined representative consensus sequences based on 20 sequences randomly picked from each cluster.

Note: Some cluster programs are capable of producing consensus sequences by themselves, this means that the relatively slow method implemented in the pipeline isn't needed. Generating a representative consensus sequence with the cluster programs is currently not implemented in the pipeline.

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 will not 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 --out_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 --out_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 --out_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 --out_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 --out_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
--out_dir comparison --pipeline cluster
```

A subset from the output will look like:

Cluster	Katwijk_ITS1.txtMaas-land_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 --out_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 `--length` setting, the length setting accepts an integer and removes all sequence reads shorter than the setting.

```
python pipeline.py --input_file fasta_file --out_dir no_small --length 200
```

This command will remove all reads from the input set that are shorter than 200 basepairs.

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 --out_dir no_dup
--duplicate yes
```

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:

```
pipeline.py [-h] [--input_file fasta files [fasta files ...]]
                  [--out_dir output directory] [--pipeline pipeline]
                  [--filter filter sequences] [--length minimum length]
                  [--duplicate remove duplicates]
                  [--trim trim input sequences]
                  [--trim_ref reference based trimming]
                  [--save_no_match save unligned seqs]
                  [--program cluster program]
                  [--similarity sequene similarity for clustering]
                  [--blast blast method]
                  [--reference reference files [reference files ...]]
                  [--accession genbank accession] [--unite unite reference]
                  [--blast_percentage minimum blast percentage]
                  [--blast_length minimum blast length]
                  [--pick_rep otu sequence picking]
                  [--min_size minimum OTU size]
                  [--rand_cons # random sequences used for the consensus]

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)
--out_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): combine (input
                      files are combined in a single file) / cluster (input
                      files are tagged and clustered together, based 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) (default: combine)
--filter filter sequences
                      Filter the sequences based on length or duplicates
                      (yes / no) default: no
--length minimum length
                      Filter out sequences smaller than the minimum length
--duplicate remove duplicates
                      Remove duplicate sequences from the dataset (yes / no)
                      default: no
--trim trim input sequences
                      Trim the input sequences yes / no (default: no)
--trim_ref reference based trimming
                      Enter the reference file for the sequence trimming (by
                      default no reference is used)
--save_no_match save unligned seqs
                      Save the sequences that cannot be aligned yes / no
                      (default: yes)
--program cluster program
                      The cluster program that will be used to cluster the
                      454 reads: uclust / cdhit / usearch (default: uclust)
--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)
--unite unite reference
                      Is the unite reference database used yes / no (if yes
                      this sets the accession parameter automatically to
                      'yes')
--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 / combined (default:
                      random)
--min_size minimum OTU size
                      minimum size for an OTU to be analyzed (default: 10)
--rand_cons # random sequences used for the consensus
                      The number of random sequences that will be pick from
                      an OTU to make a consensus sequence (default: 10)
```

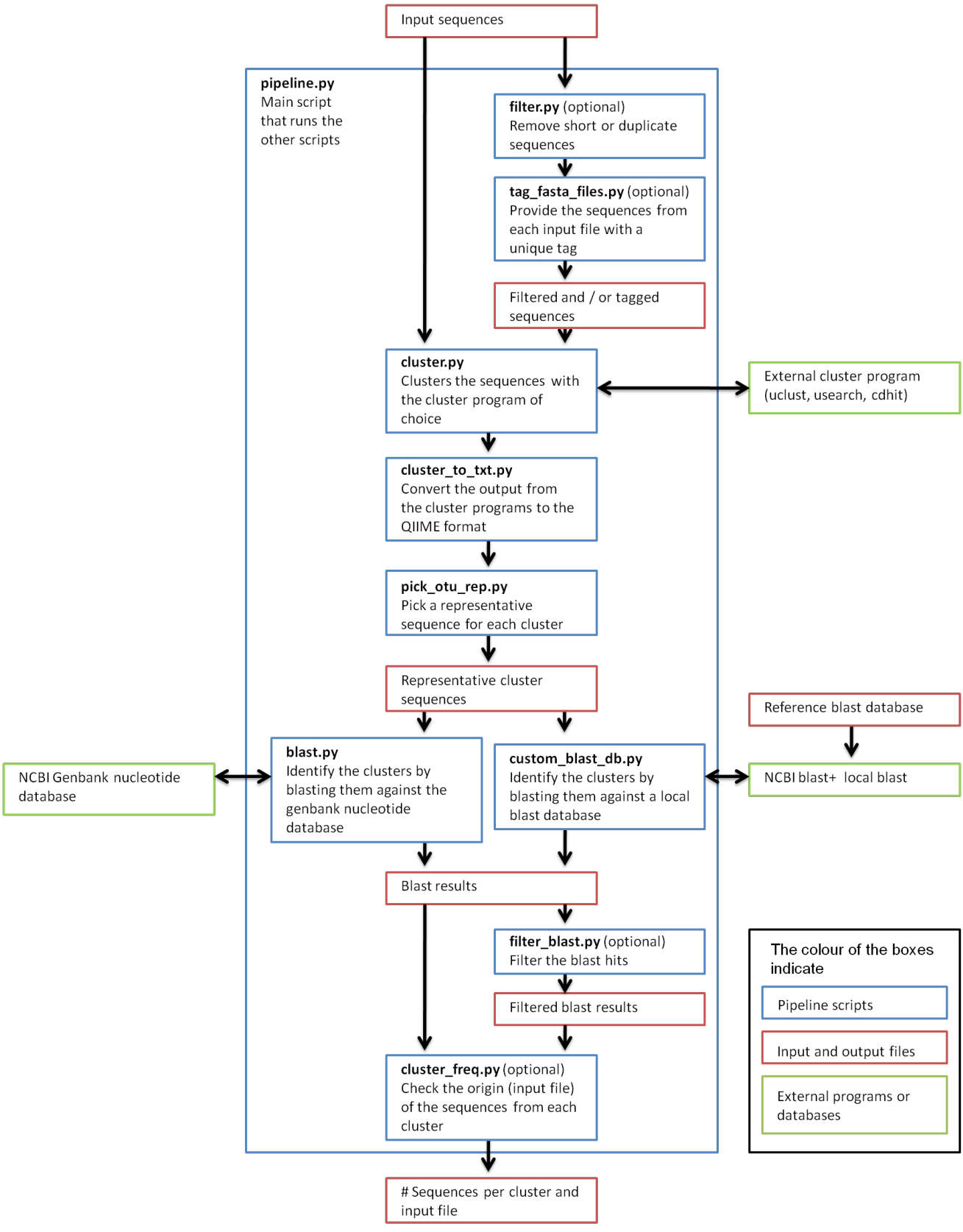

Figure 1: Flowchart of the different python scripts in the 454 cluster pipeline.