Documentation for GEMS-db pipeline

Introduction

This pipeline is a collection of scripts running different analyses for processing data amplicon-based 16S rRNA studies. Much of what is described below is now also available in pipelines such as Qiime or Mothur and the use of one of these pipelines would be better for the typical user. What is described here are the procedures we used to develop the 16S analysis pipeline for the GEMS study at a time when Qiime and Mothur did not yet exist. We also believe that the pipeline described below may be useful for educational purposes or for people who want to develop custom pipelines.

Note: This is research-grade software. Novice users beware!

All files in TAB-delimited format.
Assumes one header line - headings describe content of column
Currently does not handle 'Excel' output -need to remove quote characters

The notation convention for this document is as follows:

- Text written in fixed-width font represents UNIX command lines or file names, e.g.:
  ```
  ls -lart .
  ```
  represent the UNIX command used to list the contents of the current directory (`.`) in reverse order of access time.
- Text written in an all caps fixed-width font represents a generic name that will be replaced by the user or software with a context-dependent value, e.g.:
  ```
  dnaclust -i COMBINED.fa > COMBINED.clusters
  ```
  indicates that the generic name COMBINED will be chosen by the user. The non-capitalized part of the text is to be typed 'as is' on the command line.
- A backslash character at the end of a line indicates the command line is continued on the next line, e.g.:
  ```
  blastall -p blastn -m 8 \
           -d blastdb -o test.out
  ```
- All scripts assume 'bash' as the UNIX shell used.

The main assumption of the overall pipeline is that we can identify each sequence with a specific sample identifier, i.e. that the sample identifiers is a key in our database (different samples cannot have the same identifier). This information is either provides explicitly after demultiplexing the sequences from a same sequencing run, or is implicit in the demultiplexing process.

For demultiplexing, the current pipeline assumes that the sample identifier is a key for each run/plate. This may not be entirely correct as the experimental setup only ensures the plate-barcode combination is unique (the sequencing experiment cannot distinguish between

sequences generated using the same barcode). In general there is no reason to use the same sample ID with two different barcodes in the same run, however the user needs to doublecheck the sample information to ensure this does not happen.

Prerequisites

MySql/Postgres database
ParseTab module (currently being replaced with Text::CSV)
XML::Parser

Programs/scripts:

- dnaclust
- NCBI blast
- sff utilities (sfffile, sff_extract)

Taxonomy database

Database organized like the NCBI taxonomy database:

- `names.dmp, nodes.dmp` files linking names to taxonomy ids (`TAXID`) and taxonomy hierarchy
- fasta-formatted sequences, each with an associated database id (`DBID`)
- map between database id and associated taxonomy information (DBID `-> TAXID, TAXNAME`)

Information about samples

- [FILEIDX`.csv`] Map from sample ID (by default column name is `Sample ID`) to file information: file name, and file number. The file number is a small integer assigned to each file name in the database.
- [`CLINICAL.csv`] Map from sample ID to clinical information.
- [`BARCODES.csv`] If samples have not been demultiplexed also need map from sample ID to barcode information. This information can be represented within the `FILEIDX.csv` file rather than resorting to a separate file.

Pipeline

i. Sample demultiplexing

Each run will be placed in a single directory, containing the files listed below. Also, this directory will have sub-directories `sff/` and `seq/` for the SFF files and sequence files, respectively

Prerequisites/Inputs:

- BARCODES.csv (as described above). This file maps sample identifiers with plate location and barcode for the sequencing experiment
- Sequencer output: either a fasta file of all the sequences, or a .sff file [current pipeline works with 454 data]

Outputs:

- SAMPLEID.fa – fasta files containing all demultiplexed sequences for each sample ID
- NAMES.csv – version of the BARCODES.csv file that also includes the name of the file containing the sequences for each sample
- BAD.list, NONE.list – list of sequences that failed quality checks or did not have a recognizable barcode

i1. Demultiplex sff file

1. Create a 'mids' file (MIDS.txt) using the barcode information (can use `csv2mids.pl` for this purpose)
   Format of this file is:

```
POOL1
{
    mid = "SAMPLE_ID_1", "BARCODE1", 0;
    mid = "SAMPLE_ID_2", "BARCODE2", 0;
    ...
}

POOL2
...
```

   This file lists the mapping between the sample id and barcode in a format understandable by the 454 utilities.

2. Run `sfffile` utility from 454:

```
cd sff
sfffile -s POOL1 -mcf MIDS.txt FILE.sff
```

   The result are a collection of .sff files, one for each sample.
   Note – the POOL1 name matches the group defined in the MIDS.txt file

3. Extract sequences using the s`ff_extract` tool from 454:

   sff_extract -c -s seq/SAMPLE_ID.seq sff/SAMPLE_ID.sff

   The results are a collection of FASTA files in the seq directory.

i2. Demultiplex a FASTA file

```
demultiplexFasta.pl --in PLATE.fa --barcodes BARCODES.csv --prefix
PREFIX
```

Creates a collection of files named PREFIX.SAMPLEID.fa corresponding to each demultiplexed

sample identifier. Also creates a file named PREFIX.NONE.fa containing all sequences without a recognized barcode.

i3. Quality checking and cleanup

For each sequence in the .seq directory run the quality checking program `qc_check.pl`:

```
for i in seq/*.fa ; do
      qc_check.pl --in $i --prefix $i.clean
done
```

The output are files with suffix '`.clean.fa`' containing all good sequences as well as files with suffix '`.BAD.list`' containing all the sequences eliminated by the QC process.

i4. Create index of sample IDs to file names

```
add_file.pl --index BARCODE.csv --dir seq > NAMES.csv
add_filenum.pl NAMES.csv > NUMS.csv
```

A. Clustering

Prerequisites/Inputs:

- `FILEIDX.csv` file (as described above)
- FASTA-file containing the combined sequences from all samples [COMBINED`.fna`]. If this file does not exist it can be created with the `combinefa.pl` script as described below.

Outputs:

- `CLUSTERS.part` - XML formatted representation of clusters
- `CLUSTERS.centers.fa` - FASTA-formatted file containing cluster centers/representatives.

A.1. Combining sample sequences into a single file

```
combinefa.pl --index FILEIDX.csv --prefix COMBINED --basedir dir
```

The sequences will be output into the file called `COMBINED.fna`. They will be renamed FILENUM_`IDX` where `FILENUM` is the number assigned to the corresponding file in the index file and `IDX` is the consecutive index of the sequence in the file. The directory specified with `--basedir` is the parent directory for all files listed in the index file.

A.2. Running dnaclust

```
dnaclust -i COMBINED.fna > COMBINED.clusters
```

With default parameters dnaclust allows gaps at the end of the sequence and requires < 1%

divergence between each sequence and the cluster center. The output file (`COMBINED.clusters`) contains one line for each cluster. The lines start with the identifier of the cluster sequence and contain a space-separated list of sequences belonging to the cluster.

A.2b. Iterative dnaclust

Dnaclust can be much faster when operating in an iterative fashion: first a small subset of the sequences is clustered, all singleton clusters are discarded, then the remaining sequences are 'recruited' to the already formed clusters. The rationale for this process is to ensure that clusters containing many sequences get processed first, thereby rapidly decreasing the size of the dataset that needs to be clustered.

<TBD>

A.3. Extracting cluster centers

```
fastaselect -f COMBINED.fna -c < COMBINED.clusters >
COMBINED.centers.fa
```

A.4. Creating XML cluster representation

```
clusters2part.pl --clusters COMBINED.clusters --prefix COMBINED
```

Creates a file named `COMBINED.part` that contains the information in the input cluster file as an XML document.

NOTE: REDO FILE FORMAT

A.5. Adding more samples to already clustered collection

<TBD>

The basic idea is to use the centers file as a reference database and only use the new sequences as input to dnaclust. NEED: scripts to merge new cluster information into original partition file.

B. Assigning taxonomic information to clusters

Prerequisites:

- File containing cluster centers in fasta format (`COMBINED.centers.fa`, see Clustering for information on how to obtain this file)
- Blast-indexed 16S/18S database (`BLASTDB`)

B.1. Run Blast

```
blastall -p blastn -w 100 -d BLASTDB -i COMBINED.centers.fa -m 8 \
  -o COMBINED.blast.out
```

Or for the new NCBI Blast+:

```
blastn -word_size 100 -db BLASTDB -query COMBINED.centers.fa -outfmt 6
```

```
-out COMBINED.blast.out -max_target_seqs 1
```

The output is a standard tab-delimited BLAST format.

B.2. Create map from OTU ID to taxonomy information

```
otublasttable.pl --partition COMBINED.part|-clusters
COMBINED.otustats.csv --taxinfo TAXDB.taxinfo \
                  --blastout COMBINED.blast.out > COMBINED.taxinfo.csv
```

The inputs are the partition file created under A.4, the output of the blast command in B.1. and a file mapping the identifiers in the taxonomy database to taxonomy identifiers (`TAXDB.taxinfo`).

In addition, the location of the NCBI-formatted taxonomy database (directory `TAXDIR` containing at least the files `names.dmp` and `nodes.dmp`) can be specified with command-line option `--taxdb TAXDIR.`

The output of this process is a tab-delimited spreadsheet containing for each OTU its identifier (heading `OTU ID`), name of the center sequence (heading `Center`), taxonomy id (heading `NCBI ID`), full taxonomy (heading `NCBI taxonomy`), name of best match in taxonomy database (heading `DB ID`), and the full taxonomy as provided by the database (heading `DB taxonomy`).

C. Create count table

```
taxpart2counts.pl --part COMBINED.part --index NUM.csv --prefix NAME
```

Outputs:

- NAME.stats.txt – overall statistics about the run
- NAME.otustats.txt – per-otu statistics
- NAME.otus.count.csv – counts for each OTU (rows) and sample ID (columns) combination)

Alternative:

```
clusters2counts.pl --clusters COMBINED.clusters --index NUM.csv
--prefix NAME [--idfield <id_field>]
```

Outputs:

- NAME.otus.count.csv – rows are OTUs and columns are samples
- NAME.stats.txt – overall statistics
- NAME.otustats.csv – count statistics on a per-OTU basis

C1. Remove 'singleton' OTUs.

Simply filter the .otustats file making sure to retain the good OTUs as well has the header

"OTU_ID". This information can be provided to taxpart2summary-db.pl to restrict output just to those OTUs.

```
filter_OTUs.pl --in COMBINED.otustats.txt --prefix NAME
```

Outputs:

- NAME.otus.good.csv - list of OTUs that pass the filtering criteria

D. Normalize counts

Note: Use metagenomeSeq instead.

```
s95norm.pl --in NAME.otus.count.csv --prefix NAME
```

Outputs:

- NAME.s95.csv – table with normalized counts
- NAME.norm.csv – table with the normalization parameters (q95 and s95) for each sample.

E. Upload data to database

E1. Upload OTU count information

<TBD>

E2. Upload OTU taxonomy information

```
upload_otutaxinfo.pl
<TBD>
```

F. Create taxonomic report

```
taxpart2summary-db.pl
```

Works for both database and count table

F1. For count table

```
taxpart2summary-db.pl –prefix NAME –otutable NAME.otus.count.csv –
otutaxinfo NAME.taxinfo.csv
```

with option –filter, the output can be restricted to the good OTUs.

Running QIIME analyses

The format of the files used in our analysis is very similar to that used by Qiime [1] which enables us to use some of the analyses developed in this tool. Here are the general approach for adapting the data to the Qiime format and running these analyses.

Note: The text below is based on an older version of Qiime and may no longer work

1. Preparing sample/count information

Qiime requires the following information:

1. **Sample information** (essentially the relevant clinical information). This is a tab-delimited file containing as the first three fields:

   `#SampleID<TAB>BarcodeSequence<TB>LinkerPrimerSequence`

   The first is the sample identifier (same as `Sample ID` in our files), and the following two are just information about the sequencing experiment. As far as I can tell these appear to be required even if the samples have already been demultiplexed.

   In addition, the file has to end with another column named `Description` which can be filled in with the empty place-holder `missing_description`.

   Qiime is quite particular about the format of this file and provides a program called `check_id_map.py` that can check and fix it, if necessary.

2. **OTU count information.** This is produced by the current analysis pipeline as PREFIX`.otus.qiime.csv` (output by `taxpart2summary-db.pl`).

   Note: if the `check_id_map.py` script 'fixed' the IDs of the samples in the map file (see above) these will also have to be fixed in the count table.

3. **Qiime parameter file.** The default provided by Qiime as `qiime-parameters.txt` seems to work well.

4. **Phylogenetic tree of the OTUs** (optional). Ideally this would be created with an actual phylogenetic tree construction approach (e.g. fasttree)

2. Compute rarefaction curves/alpha-diversity analysis

The goal of this step is to compute the rarefaction curves and other ecological parameters. The commands are as follows:

```
multiple_rarefactions.py -i PREFIX.otus.qiime.csv -m 10 -x 10000 -s
492 -o RAREFACTION_DIR --num-reps 10
```

Computes bootstrapped rarefactions of the OTU table provided with option `-i` and outputs the data in directory specified with option `-o` (directory is created if not present). Options `-m` and `-x` specify the range for the boostrap and option `-s` specifies the step (data are subsampled randomly in groups of size `-m`, increasing by `-s`, up to the maximum `-x`). The `--num-reps`

option specifies how many subsamples should be created for each size - this information is used for computing confidence levels.

```
alpha_diversity.py -i RAREFACTION_DIR-o ALPHA_DIR -t None --metrics
chao1,observed_species
```

This computes a number of alpha diversity (within-sample) measures, such as chao/ace diversity measures, simpson, shannon, etc. Option `-t` specifies an optional phylogenetic tree for phylogenetic measures. Option `-s` specifies all available measures, and option `-o` specifies where the data should be output.

```
collate_alpha.py -i ALPHA_DIR -o ALPHA_COLLATED
```

Collates the alpha diversity information in preparation for plotting

```
make_rarefaction_plots.py -i ALPHA_COLLATED -m CLINICAL_MAP.csv -o
ALPHA_PLOTS --background_color white --resolution 75 --imagetype png
```

This draws a collection of pretty plots and creates an HTML interface to these plots. Parameter `-m` specifies the clinical map information allowing the plots to be investigated on the basis of clinical parameters.

3. Multi-dimensional scaling analysis (Beta-diversity - between-sample diversity)

This is the typical analysis for which Qiime is famous.

```
beta_diversity.py -i PREFIX.otus.qiime.csv -t PREFIX.otus.tre
  -o OUTDIR -m unweighted_unifrac,weighted_unifrac,euclidean
```

This is the main part that computes a distance matrix on the basis of one or more definitions of distance (specified with the `-m` option)

```
principal_coordinates.py -i euclidean_PREFIX.otus.qiime.csv -o PLOTDIR
```

Each distance matrix is converted into multi-dimensional scaling coordinates (principal coordinates) and the information is output in the directory specified by option `-o`.

```
make_3d_plots.py -i PLOTDIR -m CLINICAL_MAP.csv
        -b HIV,Source,Site,ART,Smoker -p qiime_parameters.txt
        -o GRAPHDIR
```

This command produces the pretty interactive 3D plots, creating differently colored plots based on the clinical parameters listed in option `-b`. The resulting HTML documents can be viewed in most modern browsers.

References

1. Caporaso, J.G., et al., *QIIME allows analysis of high-throughput community sequencing data.* Nat Methods, 2010. 7(5):335-6.