THAPBI PICT

Release 1.0.12

Peter Cock

Mar 11, 2024

DOCUMENTATION CONTENTS:

1 Introduction 3

2 Installation 7

3 Quick Start 9

4 Worked Examples 15

5 Reference database 135

6 Abundance & Negative Controls 137

7 Classifier Assessment 139

8 Command Line 141

9 Python API 143

10 Release History 165

11 Development Notes 169

Python Module Index 173

Index 175

THAPBI PICT is a sequence based diagnostic/profiling tool from the UK funded Tree Health and Plant Biosecurity Initiative (THAPBI) Phyto-Threats project, initially focused on identifying *Phytophthora* species present in Illumina sequenced environmental samples.

Phytophthora (from Greek meaning plant-destroyer) species are economically important plant pathogens, in both agriculture and forestry. ITS1 is short for Internal Transcribed Spacer one, which is a region of eukaryotes genomes between the 18S and 5.8S rRNA genes. This is commonly used for molecular barcoding, where sequencing this short region can identify species.

With appropriate primer settings and a custom database of full length markers, THAPBI PICT can be applied to other organisms and/or barcode marker sequences - not just *Phytophthora* ITS1. It requires overlapping paired-end Illumina reads which can be merged to cover the *full* amplicon marker. Longer markers or fragmented amplicons are not supported. Internally it works by tracking unique amplicon sequence variants (ASVs), using MD5 checksums as identifiers.

The worked examples include oomycetes, fungi, fish, bats, and plants, and cover markers in ITS1, ITS2, 12S, 16S, COI, and more. The main criteria has been mock communities with known species composition.

The THAPBI Phyto-Threats project was initially supported by a grant funded jointly by the Biotechnology and Biological Sciences Research Council (BBSRC), the Department for Environment, Food and Rural affairs (DEFRA), the Economic and Social Research Council (ESRC), the Forestry Commission, the Natural Environment Research Council (NERC) and the Scottish Government, under the Tree Health and Plant Biosecurity Initiative (THAPBI).

Key links:

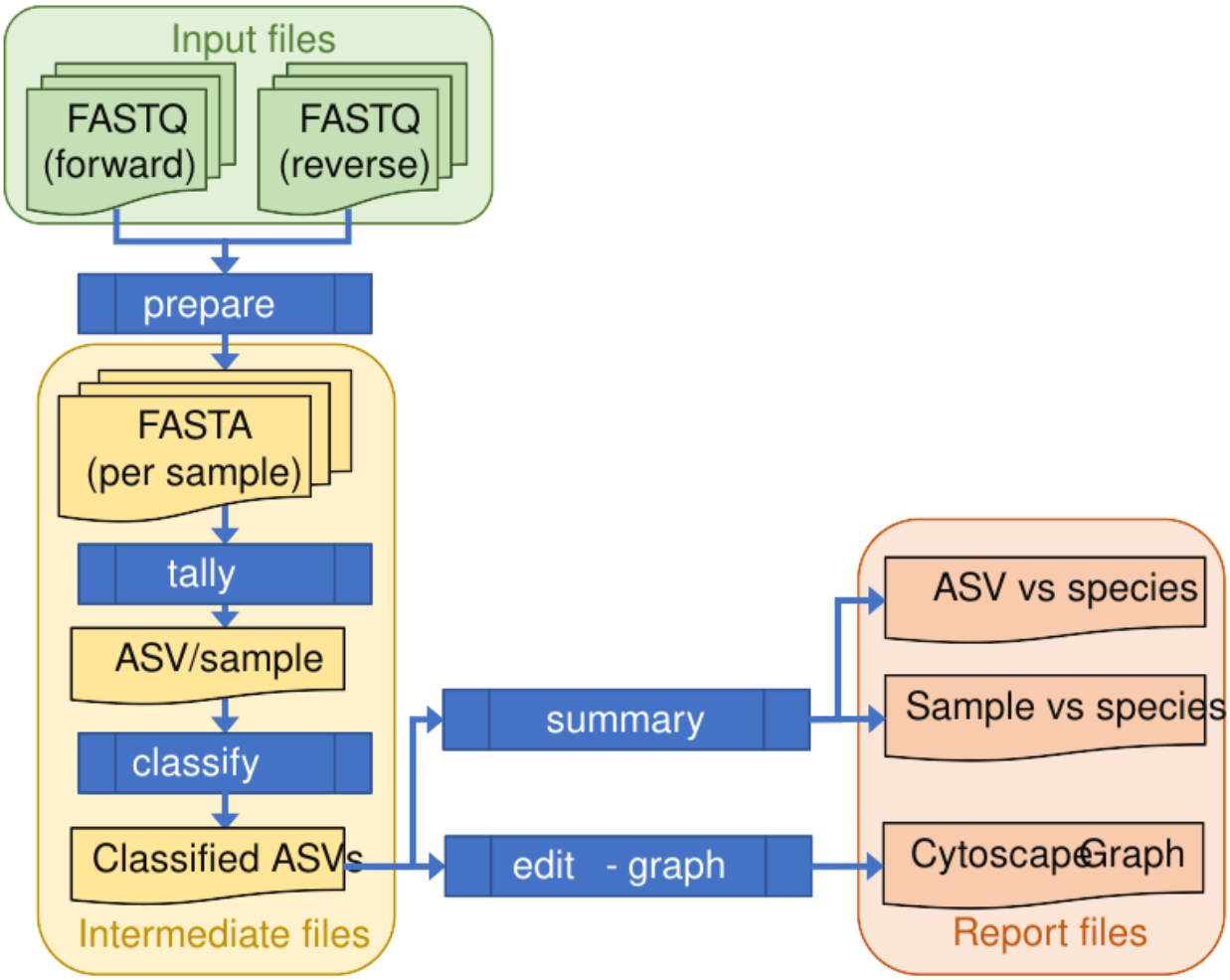
- Documentation on Read The Docs: https://thapbi-pict.readthedocs.io/
- Source code repository on GitHub: https://github.com/peterjc/thapbi-pict/
- Software released on PyPI: https://pypi.org/project/thapbi-pict/
- Zenodo DOI for software: https://doi.org/10.5281/zenodo.4529395
- Paper on PeerJ: https://doi.org/10.7717/peerj.15648

CHAPTER

ONE

INTRODUCTION

THAPBI PICT is a tool designed to assess species content of metabarcode amplicons sequenced using an overlapping paired-end Illumina protocol. The input data is paired FASTQ files (one pair for each sample), from which unique sequences (commonly called unique amplicon sequence variants, ASVs) and their abundance are reported alongside one or more matching species or genus names.



In this illustrative flow chart of the default pipeline, the input paired FASTQ files are green, the intermediate per-sample FASTA and TSV files are yellow, and the output reports are in orange. The individual steps of the pipeline are dark blue boxes.

1.1 Read preparation

The first stage of the pipeline goes from a set of paired FASTQ files to a set of non-redundant primer trimmed FASTA files and sample tallies per marker. This currently runs as follows:

1. Merge overlappping reads into single sequences, using Flash (Magoc and Salzberg 2011).
2. Filter for primers and trim to target region, using Cutadapt (Martin 2011).
3. Tally unique sequences per sample per marker.
4. Optionally apply UNOISE read correction (de-noising).
5. Apply a minimum abundance threshold (guided by any negative controls).

1.2 Sequence classification

The second stage of the pipeline offers a choice of classifier algorithms:

- 100% identity (`identity`). Requires the primer trimmed read sequence match a database entry exactly. The database entries must be trimmed too.
- Up to one base pair away (`onebp`, the *default*). Like the identity classifier, but allows a single base pair edit (a substitution, deletion, or insertion).
- Up to one base pair away for a species level match (like the default `onebp` method), but falling back on up to 2bp, 3bp, 4bp, ... away for a genus level match (`1s2g`, `1s3g`, `1s4g`, ...).
- Perfect substring (`substr`). Like the identity classifier, but also allows for the query sequence to be a perfect substring of a database entry. Useful if the database entries have not all been trimmed exactly.
- Top BLAST hit within database (`blast`). This classifier calls NCBI BLAST with a local database built of the database sequences, and takes the species of the top BLAST hit(s) subject to some minimum alignment quality to try to exclude misleading matches.

These have different strengths and weaknesses, which depend in part on the completeness of the database for the target environment. The `identity`, `substr` and `onebp` classifiers are very strict, and with a sparse database could leave a lot a lot of sequences with no prediction. On the other hand, the `blast` classifier is much more fuzzy and will make classifications on much looser criteria - but with a sparse database those matches could easily be false positives.

In assessing the classification performance, it is the *combination* of both classification method (algorithm) *and* marker database which which matters.

1.3 Classification output

The classifier output is at unique sequence level, reporting zero or more species matches (or genus matches from some classifiers, or if sequences in the database are recorded at genus level only).

For example, an ITS1 sequence from a known *Phytophthora infestans* single isolate control can in addition to this expected result also perfectly match sister species *P. andina* and *P. ipomoeae*. Here the classifier would report all three species (sorted alphabetically), giving:

```
Phytophthora andina;Phytophthora infestans;Phytophthora ipomoeae
```

If additionally the query sequence matched genus level only *Phytophthora* entries in the database, that would be redundant information, and not reported in this example.

Neither in this raw classification output, nor the provided reports, does THAPBI PICT currently attempt any simiplification like last reporting the common ancestor of a complex result. For the initial use case focused on *Phytophthora*, this is simply not needed.

1.4 Reporting

There are currently three main reports produced (in multiple formats including formatted Excel spreadsheets).

- **Sample report**. Table with samples as rows, and genus and species as columns, with combined sequence counts as values. Includes a total row, and unclassified counts as additional column. Can include sample metadata as additional columns.
- **Read report**. Table of unique sequences as rows, and samples as columns, with read counts (sequence abundance) as values. Includes any species classification and the sequences themselves as additional columns. Can include sample metadata as additional header rows.
- **Edit graph**. Represents all the unique sequences in the sample (plus optionally all those in the reference database) as nodes with edges between them for edit distance (solid black lines for 1bp, dashed grey for 2bp, and dotted grey for 3bp away). Any sequences also in the database are colored.

These are discussed and excerpts shown in the *worked examples* later in the documentation.

CHAPTER

TWO

INSTALLATION

2.1 First time installation

We recommend installing this tool on Linux or macOS using the Conda packaging system, via the BioConda channel, which will handle *all* the dependencies:

```
$ conda install thapbi-pict
```

Alternatively, since the software is on the Python Package Index (PyPI), the following command will install it along with its Python dependencies:

```
$ pip install thapbi-pict
```

However, in this case you will still need to install at least the command line tool `flash` (for merging Illumina paired reads), and optionally others like NCBI BLAST+ (used for one classifier method). If you have BioConda setup, use the following:

```
$ conda install --file requirements-ext.txt
```

If you are not using Conda, then on a typical Linux system most of the tools required will be available via the default distribution packages, although not always under the same package name.

On Debian (with the efforts of DebianMed), or Ubuntu Linux, try:

```
$ sudo apt-get install ncbi-blast+
```

If you are on Windows, and do not wish to or cannot use the Windows Subsystem for Linux (WSL), the tool can be installed with `pip`, but you will have to manually install the command line dependencies. Download a pre-compiled binary from https://ccb.jhu.edu/software/FLASH/ and BLAST+ (if required) from the NCBI, and ensure they are on the system PATH. To run the test suite and worked example scripts, you will also need a bash shell with basic Unix tools like grep.

If you want to install the very latest unreleased code, you must download the source code from the repository on GitHub - see the `CONTRIBUTING.rst` file for more details.

Once installed, you should be able to run the tool using:

```
$ thapbi_pict
```

This should automatically find the installed copy of the Python code. Use `thapbi_pict -v` to report the version, or `thapbi_pict -h` for help.

2.2 Updating

If you installed via conda, this should work:

```
$ conda update thapbi-pict
```

If you installed via pip, this should work:

```
$ pip install --upgrade thapbi-pict
```

Either way, you can check the installed tool version using:

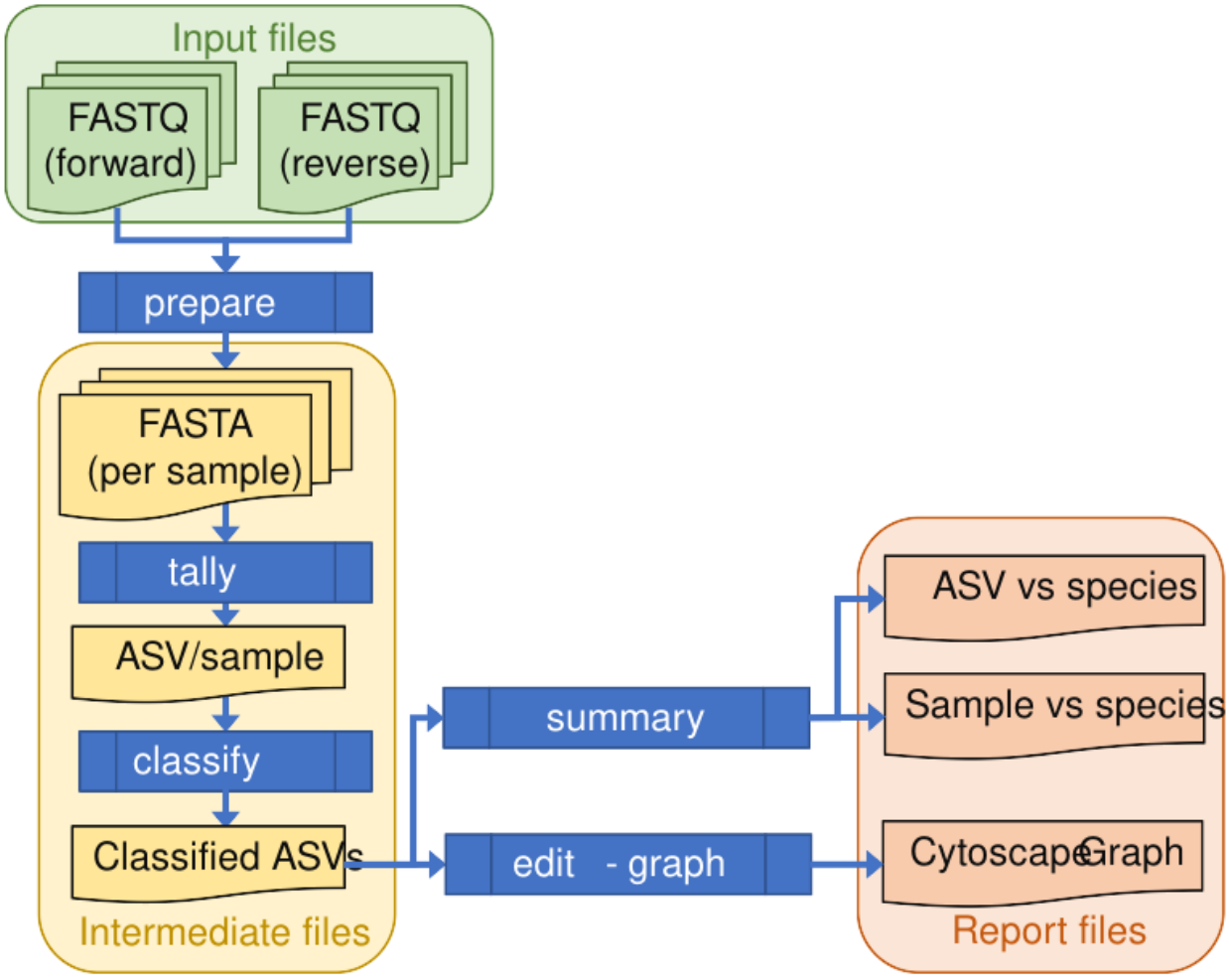
```
$ thapbi_pict -v
```

CHAPTER

THREE

QUICK START

Here we describe a simplified use of the THAPBI PICT tool to assess a single Illumina MiSeq sequencing run. The input data is a set of paired FASTQ files (one pair for each sample), perhaps barcoded samples from a 96-well plate.



In this illustrative flow chart of the default pipeline, the input paired FASTQ files are green, the intermediate per-sample FASTA and TSV files are yellow, and the output reports are in orange. The individual steps of the pipeline are dark blue boxes.

We will now describe how to run the `thapbi_pict pipeline` command, which will process the samples, make classifications, and summary reports.

```
$ thapbi_pict pipeline -h
...
```

3.1 Setup

We assume you have a new folder dedicated to this analysis, with a sub folder `raw_data/` which contains the demultiplexed paired FASTQ files which are named like `<sample_name>_R1.fastq.gz` and `<sample_name>_R2.fastq.gz` as provided by your sequencing centre. The tool understands a few widely used naming patterns. We recommend that you *do not* decompress the FASTQ files (as `<sample_name>_R1.fastq` and `<sample_name>_R2.fastq`), leaving them gzip compressed is preferable for disk space.

```
$ cd /path/to/my/project/
$ ls raw_data/*.fastq.gz
...
```

We will make two additional sub-folders, `intermediate/` (for the per-sample prepared FASTA files), and `summary/` for the folder level reports.

```
$ mkdir -p intermediate/ summary/
```

3.2 Running

With that done, we run the `thapbi_pict pipeline` command, which for a single 96 sample Illumina MiSeq run should take a minute or so:

```
$ thapbi_pict pipeline -i raw_data/ -s intermediate/ -o summary/thapbi-pict
...
Wrote summary/thapbi-pict.ITS1.samples.onebp.*
Wrote summary/thapbi-pict.ITS1.reads.onebp.*
All done!
```

This is robust to being interrupted and restarted (as long as you are not changing settings), and will reuse intermediate files:

```
$ thapbi_pict pipeline  -i raw_data/ -s intermediate/ -o summary/thapbi-pict
...
Skipped 120 previously prepared ITS1 samples
...
Wrote summary/thapbi-pict.ITS1.samples.onebp.*
Wrote summary/thapbi-pict.ITS1.reads.onebp.*
All done!
```

All being well, this will produce a set of report files, with names starting with the prefix `summary/thapbi-pict.*` given as follows:

```
$ ls -1 summary/thapbi-pict.*
summary/thapbi-pict.ITS1.onebp.tsv
summary/thapbi-pict.ITS1.reads.onebp.tsv
summary/thapbi-pict.ITS1.reads.onebp.xlsx
```

(continues on next page)

(continued from previous page)

```
summary/thapbi-pict.ITS1.samples.onebp.tsv
summary/thapbi-pict.ITS1.samples.onebp.xlsx
summary/thapbi-pict.ITS1.tally.tsv
```

Warning: This minimal example omits a key consideration - telling the tool which samples are negative controls, and/or manually setting the minimum read abundance.

3.3 Intermediate FASTA files

The first stage of the pipeline can be run separately as the `thapbi_pict prepare` command. Here each pair of FASTQ files named something like `<sample_name>_R1.fastq.gz` and `<sample_name>_R2.fastq.gz` is processed to give a much smaller FASTA format files `<marker_name>/<sample_name>.fasta` for each marker, containing all the unique sequences from that sample which have the expected primers (so here should resemble an ITS1 sequence or our synthetic controls).

In these FASTA files, each sequence is named as `<checksum>_<abundance>` where the MD5 checksum of the sequence and is used as a unique shorthand - a 32 character string of the digits `0` to `9` and lower cases letters `a` to `f` inclusive. These MD5 checksums are used later in the pipeline, including in the read reports.

The intermediate FASTA files start with a header made of multiple lines starting with `#`, which records information about the sample for use in reporting. This includes which marker this was and the primers, how many raw reads the FASTQ files had, how many were left after pair merging, and primer trimming. Many third-party tools will accept these files as FASTA without complaint, but some tools require the header be removed.

The second stage of the pipeline can be run separately as the `thapbi_pict sample-tally` command. This produces a sequence versus sample tally table as a tab-separated table (TSV file), with the sequences as the final column. This is file `summary/thapbi-pict.ITS1.tally.tsv` in the above example.

This step can optionally produce a pooled non-redundant FASTA file with all the observed marker sequences in it (and the total read abundance).

3.4 Intermediate TSV files

The third stage of the pipeline can be run separately as the `thapbi_pict classify` command. Here species predictions are made for each sequence in the prepared sequence vs sample tally file, generating a TSV file where the first column is the sequence name in `<checksum>_<abundance>` format. This is file `summary/thapbi-pict.ITS1.onebp.tsv` in the above example.

3.5 Sample Reports

The first set of reports from the pipeline or `thapbi_pict summary` command are the sample reports - using the filenames from the above example:

- Plain table `summary/thapbi-pict.ITS1.samples.onebp.tsv` (tab separated variables, TSV) which can be opened in R, Excel, or similar.
- Visually formatted table `summary/thapbi-pict.ITS1.samples.onebp.xlsx` (Microsoft Excel format), with the same content but with colors etc applied.

These aim to give a summary of the species identified within each sample. The tables have one row for each sample. The main columns give total read counts, those not matched to anything ("Unknown"), reads matched at species level (with ambiguous combinations listed explicitly), and reads matched only to genus level.

In the Excel version, conditional formatting is used to highlight the non-zero counts with a red background.

3.6 Read Reports

The other report from the pipeline or `thapbi_pict summary` command is more detailed, being at the level of the unique sequences or reads. Again using the filenames from the above example:

- Plain table `summary/thapbi-pict.ITS1.reads.onebp.tsv` (tab separated variables, TSV) which can be opened in R, Excel, or similar.
- Visually formatted table `summary/thapbi-pict.ITS1.reads.onebp.xlsx` (Microsoft Excel format), with the same content but with colors etc applied.

This read report has a row for each unique sequence. The first columns are the unique sequence MD5 checksum, any species prediction, the sequence itself, the number of samples it was detected in above the threshold, and the total number of times this was seen (in samples where it was above the threshold). Then the main columns (one per sample) list the abundance of each unique sequence in that sample (if above the threshold).

In the Excel version, conditional formatting is used to highlight the non-zero counts with a red background.

3.7 Edit Graph

While not run by the pipeline, there is a separate `thapbi_pict edit-graph` command, where the default output is:

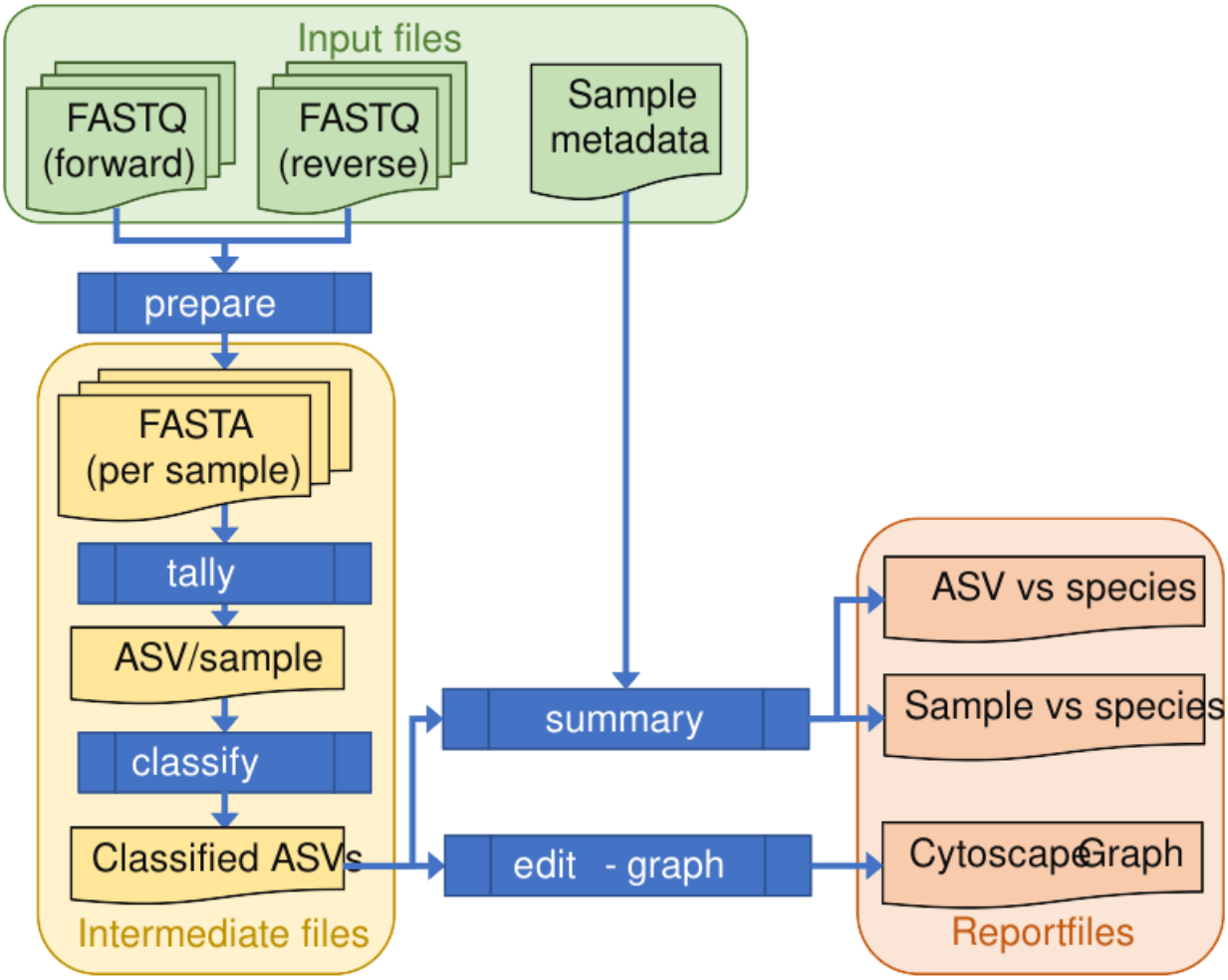
- Edit-distance graph `XXX.edit-graph.xgmml` (XGMML, eXtensible Graph Markup and Modeling Language) which we recommend opening in Cytoscape.

Note that `thapbi_pict edit-graph` supports other node-and-edge graph file formats, and can produce a static PDF image as well using GraphViz and other dependencies, or a distance matrix.

3.8 Next Steps

This minimal example omits a key consideration which is telling the tool which of the samples are your negative controls and/or manually setting the minimum read abundance.

Also, interpreting the main reports is much easier if you can provide suitably formatted *metadata*. Happily, you can quickly re-run the pipeline and it will reuse any already generated intermediate files.



The *first worked example* covers these issues, with excerpts of the expected output.

CHAPTER

FOUR

WORKED EXAMPLES

While THAPBI PICT stands for *Phytophthora* ITS1 Classification Tool, with appropriate primer settings and a custom database, it can be applied to other organisms and/or barcode marker sequences.

These worked examples use public datasets from published papers, with various markers covering oomycetes, fungi, animals and plants. The main criteria has been mock communities with known species composition.

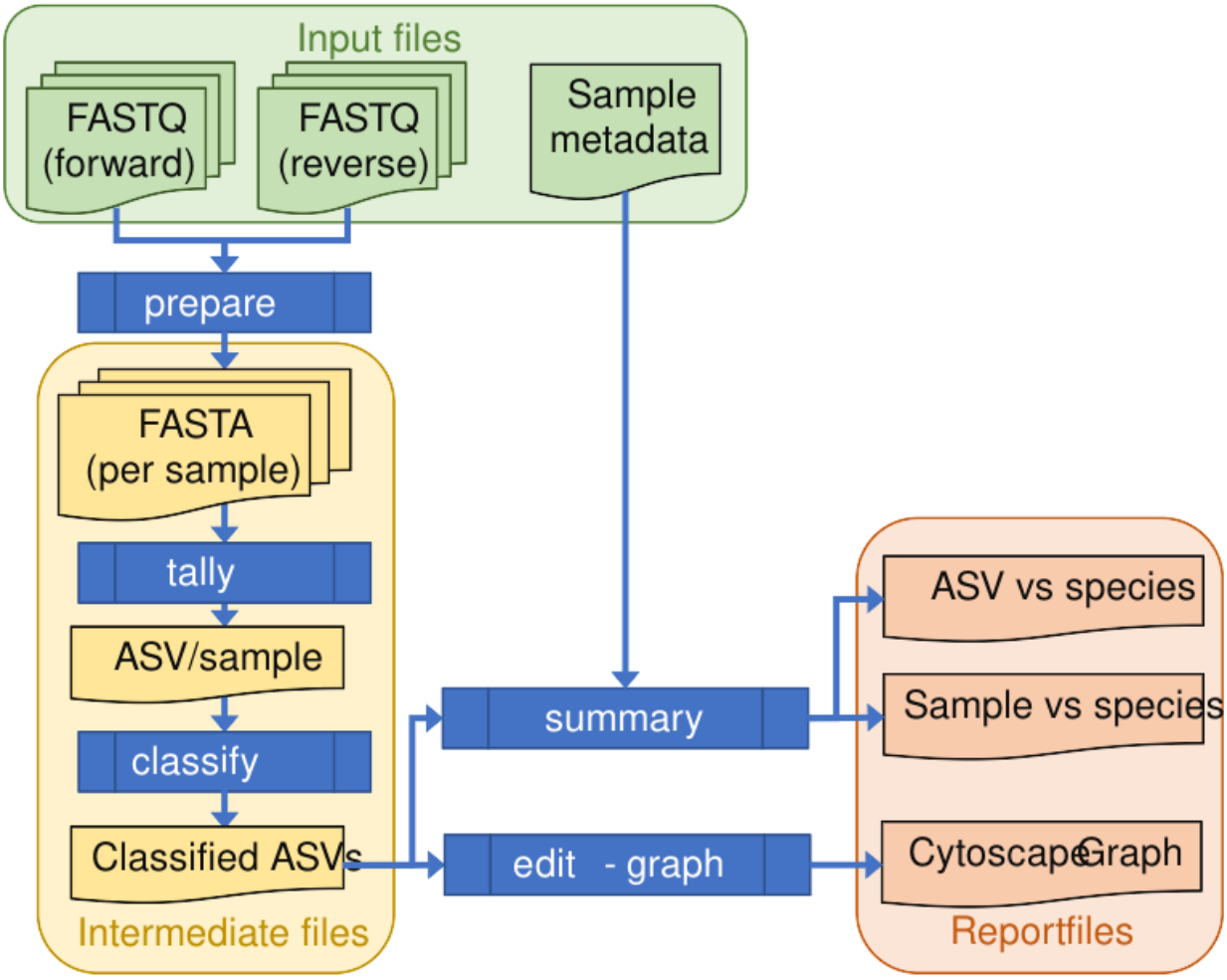
4.1 Environmental *Phytophthora* ITS1

This example is based on the following paper from earlier in the THAPBI Phyto-Threats project, where the original analysis used the precursor pipeline `metapy`:

> Riddell *et al.* (2019) Metabarcoding reveals a high diversity of woody host-associated *Phytophthora* spp. in soils at public gardens and amenity woodlands in Britain. https://doi.org/10.7717/peerj.6931

Importantly, they used the same PCR primers, and therefore analysis with this tool's default settings including the provided database is appropriate.

The *Quick Start* described a simplified use of the THAPBI PICT tool to assess a single Illumina MiSeq sequencing run using the `thapbi_pict pipeline` command, as a flowchart:



Here we will run over the same process using real *Phytophthora* ITS1 data, calling the individual commands within the default pipeline - and include metadata for reporting. We then run the equivalent all-in-one pipeline command.

Finally, since the sample data includes some positive controls, we can look at assessing the classifier performance.

4.1.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/woody_hosts/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

The documentation goes through running each step of the analysis gradually, before finally calling pipeline command to do it all together. We provide script `run.sh` to do the final run-though automatically (first without any metadata, then again with it), but encourage you to follow along the individual steps first.

FASTQ data

The raw data is from two Illumina MiSeq runs, a whole 96-well plate from 2016, and about half the samples from a second 96-well plate sequenced in 2017 (where the rest of the plate was samples from a separate ITS1 study). There are multiple replicates from each of 14 sample sites, plus controls. The raw FASTQ files are too large to include with the THAPBI PICT source code.

Script `setup.sh` will download the raw FASTQ files for Riddell *et al.* (2019) from https://doi.org/10.5281/zenodo.3342957

It will download 244 raw FASTQ files (122 pairs), about 215MB on disk

Amplicon primers & reference sequences

The ITS1 primers used here match the THAPBI PICT defaults, so the default database can also be used.

Metadata

The provided file `metadata.tsv` is an expanded version of Supplementary Table 1 from the original paper, adding a column for the Illumina MiSeq sample names, rows for the controls.

The 16 columns are as follows, where 4 to 15 are in pairs for tree/shrub broad taxonomic grouping and health status (H, healthy; D, symptoms/stump/dead):

1. Anonymised site number (with leading zero, "01" to "14"), or control name
2. Approximate altitude at centre
3. Underlying soil type
4. Healthy Cupressaceae
5. Diseased Cupressaceae
6. Healthy other conifers
7. Diseased other conifers
8. Healthy Ericaceae
9. Diseased Ericaceae
10. Healthy Fagaceae or Nothofagaceae
11. Diseased Fagaceae or Nothofagaceae
12. Healthy other angiosperms
13. Diseased other angiosperms
14. Healthy other
15. Diseased other
16. MiSeq Sample(s) (semi-colon separated list)

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 16 -c 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
```

These settings are described in detail later (see *Metadata*). This example is important in that column 16 contains multiple entries where a site had multiple sequenced samples (replicates).

Other files

Subdirectory `expected/` contains four plain text tab-separated files, describing the expected species in some mock community positive controls:

- `DNA15MIX.known.tsv`
- `DNA10MIX_bycopynumber.known.tsv`
- `DNA10MIX_diluted25x.known.tsv`
- `DNA10MIX_undiluted.known.tsv`

4.1.2 Preparing the sequence data

Running thapbi-pict prepare-reads

Calling `thapbi-pict prepare-reads` is the first action done by the top level `thapbi_pict pipeline` command.

```
$ thapbi_pict prepare-reads -h
...
```

Assuming you have the FASTQ files in `raw_data/` as described above:

```
$ thapbi_pict prepare-reads -i raw_data/ -o intermediate/
...
```

For each input FASTQ file pair `raw_data/<sample_name>_R1.fastq.gz` and `raw_data/<sample_name>_R2.fastq.gz` you should get a small FASTA file `intermediate/<marker_name>/<sample_name>.fasta`. In this case, there are multiple replicates from each of 14 sample sites where the file name stem is `Site_<N>_sample_<X>`, plus the controls.

```
$ ls -1 intermediate/ITS1/*.fasta | wc -l
122
```

Note this is robust to being interrupted and restarted (e.g. a job might time out on a cluster).

You should find 122 small FASTA files in the `intermediate/ITS1/` folder

Note that four of these FASTA files are empty, `Site_13_sample_7.fasta` and `Site_9_sample_4-3.fasta` (nothing above the minimum threshold), and both negative controls (good).

Warning: So far this example omits a key consideration - telling the tool which samples are negative controls, and/or manually setting the minimum read abundance. See below.

Intermediate FASTA files

What the prepare command does can be briefly summarised as follows:

- Merge the overlapping paired FASTQ reads into single sequences (pairs which do not overlap are discarded, for example from unexpectedly long fragments, or not enough left after quality trimming).
- Primer trim (reads without both primers are discarded).
- Convert into a non-redundant FASTA file, with the sequence name recording the abundance (discarding sequences of low abundance).
- If synthetic controls are defined in the DB, look for matches using k-mers. These will be discounted when using negative control samples to raise the minimum abundance threshold for the plate.

For each input `<sample_name>_R1.fastq.gz` and `<sample_name>_R2.fastq.gz` FASTQ pair we get a single *much* smaller FASTA file `<sample_name>.fasta`.

Warning: The intermediate FASTA files can legitimately have no sequences which passed the thresholds. This can happen when a PCR failed, and is expected to happen on blank negative controls.

Warning: The intermediate FASTA files start with an atypical header made up of lines starting `#`. Some tools need this to be removed, but others will accept this as valid FASTA format.

For example, here the header tells us this sample started with 6136 reads in the paired FASTQ files, down to just 4180 after processing (with the final step being the abundance threshold).

```
$ head -n 12 intermediate/ITS1/Site_1_sample_1.fasta
#marker:ITS1
#left_primer:GAAGGTGAAGTCGTAACAAGG
#right_primer:GCARRGACTTTCGTCCCYRC
#threshold_pool:raw_data
#raw_fastq:6136
#flash:5900
#cutadapt:5886
#abundance:5194
#threshold:2
#singletons:692
>2e4f0ed53888ed39a2aee6d6d8e02206_2269
TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAAACTTTCCACGTGAACTGTATCGAACAACTAGTTGG
GGGTCTTGTTTGGCGTGCGGCTGCTTCGGTAGCTGCTGCTAGGCGAGCCCTATCACGGCGAGCGTTTGGACTTCGGTCTG
AGCTAGTAGCTATTTTTTAAACCCATTCTTTAATACTGATTATACT
```

The sequence entries in the FASTA file are named `<checksum>_<abundance>`. Here `<checksum>` is the MD5 checksum of the sequence, and this is used as a unique shorthand. It is a 32 character string of the digits `0` to `9` and lower cases letters `a` to `f` inclusive, like `a559aa4d00a28f11b83012e762391259`. These MD5 checksums are used later in the pipeline, including in reports. The `<abundance>` is just an integer, the number of paired reads which after processing had this unique sequence.

Any description entry in the FASTA records after the identifier is the name of the synthetic spike-in sequence in the database that was matched to using *k*-mer counting (so `2e4f0ed53888ed39a2aee6d6d8e02206_2269` was not a spike-in sequence).

The order of the FASTA sequences is in decreasing abundance, so the first sequence shown

`2e4f0ed53888ed39a2aee6d6d8e02206_2269` is the most common, and so that read count 2269 also appears in the headers as the maximum non-spike-in abundance (with no spike-in reads in this sample).

Note the sequence in the FASTA file is written as a single line in upper case. With standard FASTA line wrapping at 60 or 80 characters, the ITS1 sequences would need a few lines each. However, they are still short enough that having them on one line without line breaks is no hardship - and it is *extremely* helpful for simple tasks like using `grep` to look for a particular sequence fragment at the command line.

Note that for this documentation, the FASTA output has had the sequences line wrapped at 80 characters.

```
$ grep "^>" intermediate/ITS1/Site_1_sample_1.fasta | head -n 8
>2e4f0ed53888ed39a2aee6d6d8e02206_2269
>c1a720b2005f101a9858107545726123_715
>96e0e2f0475bd1617a4b05e778bb04c9_330
>fb30156d7f66c8abf91f9da230f4d19e_212
>dcd6316eb77be50ee344fbeca6e005c7_194
>972db44c016a166de86a2bacab3f4226_182
>d9bc3879fdab3b4184c04bfbb5cf6afb_165
>ed15fefb7a3655147115fc28a8d6d671_113
```

The final output has just eight unique sequences accepted, happily none of which match the synthetic controls. The most common is listed first, and had MD5 checksum `2e4f0ed53888ed39a2aee6d6d8e02206` and was seen in 2269 reads.

You could easily find out which other samples had this unique sequence using the command line search tool `grep` as follows:

```
$ grep 2e4f0ed53888ed39a2aee6d6d8e02206 intermediate/*.fasta
...
```

Or, since we deliberately record the sequences without line wrapping, you could use `grep` with the actual sequence instead (which might spot some slightly longer entries as well).

You can also answer this example question from the read report produced later.

Abundance thresholds

As you might gather from reading the command line help, there are two settings to do with the minimum read absolute abundance threshold, `-a` or `--abundance` (default 100), and `-n` or `--negctrls` for specifying negative controls (default none).

(See also *Abundance & Negative Controls* which discusses the use of the fractional abundance threshold `-f` or `--abundance-fraction` and how to set this dynamically with synthetic control samples with `-y` or `--synthetic`.)

If any negative controls are specified, those paired FASTQ files are processed *first*. If any of these contained ITS1 sequences above the specified minimum absolute abundance threshold (default 100), that higher number is used as the minimum abundance threshold for the non-control samples. For example, say one control had several ITS1 sequences with a maximum abundance of 124, and another control had a maximum ITS1 abundance of 217, while the remaining controls had no ITS1 sequence above the default level. In that case, the tool would take maximum 217 as the abundance threshold for the non-control samples.

If you wished to lower the threshold from the default to 50, you could use:

```
$ rm -rf intermediate/ITS1/*.fasta  # Are you sure?
$ thapbi_pict prepare-reads -i raw_data/ -o intermediate/ -a 50
...
```

Warning: By default `thapbi_pict prepare-reads` and `thapbi_pict pipeline` will reuse existing intermediate FASTA files, so you must explicitly delete any old FASTA files before the new abundance threshold will have any effect.

Warning: Setting the abundance threshold low (say under 50) risks background contamination coming through into the results. Do not do this without strong justification (e.g. look at suitable controls over multiple plates from your own laboratory procedure).

Warning: Setting the abundance threshold *very* low (under 10) has the additional problem that the number of unique sequences accepted will increase many times over. This will *dramatically* slow down the rest of the analysis. This is only advised for investigating single samples.

For the woody host data, each plate had a negative control sample which should contain no ITS1 sequences. We can specify the negative controls with `-n` or `--negctrls` by entering the four FASTQ filenames in full, but since they have a common prefix we can use a simple wildcard:

```
$ thapbi_pict prepare-reads -i raw_data/ -o intermediate/ -n raw_data/NEGATIVE*.fastq.gz
...
```

For this sample data, happily neither of the negative controls have any ITS1 present above the default threshold, so this would have no effect.

For the THAPBI Phyto-Threats project we now run each 96-well PCR plate with multiple negative controls. Rather than a simple blank, these include a known mixture of synthetic sequences of the same length, same nucelotide composition, and also same di-nucleotide composition as real *Phytophthora* ITS1. This means we might have say 90 biological samples which should contain ITS1 but not the synthetics controls, and 6 negative controls which should contain synthetic controls but not ITS1.

We therefore run `thapbi_pict prepare-reads` separately for each plate, where any ITS1 contamination in the synthetic controls is used to set a plate specific minimum abundance. This means we cannot run `thapbi_pict pipeline` on multiple plates at once (although we could run it on each plate, we generally want to produce reports over multiple plates).

4.1.3 Classifying sequences

Running thapbi-pict classify

The second stage of the pipeline is to merge all the sample specific FASTA files into one non-redundant sequence vs sample TSV file, ready to classify all the unique sequences in it. These steps can be run separately:

```
$ thapbi_pict sample-tally -h
...
$ thapbi_pict classify -h
...
```

There are a number of options here, but for the purpose of this worked example we will stick with the defaults and tell it to look for FASTA files in the `intermediate/` directory.

```
$ thapbi_pict sample-tally -i intermediate/ITS1/*.fasta -o summary/thapbi-pict.ITS1.
↪tally.tsv
...
$ thapbi_pict classify -i summary/thapbi-pict.ITS1.tally.tsv
...
```

Here we have not set the output folder with `-o` or `--output`, which means the classify step will default to writing the classifier TSV output file next to the input tally TSV file. There should now be two new files:

```
$ ls -1 summary/thapbi-pict.ITS1.*.tsv
summary/thapbi-pict.ITS1.onebp.tsv
summary/thapbi-pict.ITS1.tally.tsv
```

If you have the `biom-format` Python library installed, adding `--biom` to the command line will result in a `summary/thapbi-pict.ITS1.onebp.biom` file as well, equivalent to the data in `summary/thapbi-pict.ITS1.tally.tsv` but potentially more useful for export to other analysis tools.

Intermediate TSV files

For each input tally TSV file `<name>.tally.tsv` another plain text TSV file is generated named `<name>.<method>.tsv` where the default method is `onebp` (which looks for perfect matches or up to one base pair different). These are both sequence versus sample observation tables of counts, but with sample metadata in header lines (starting with #) and additional columns for the amplicon marker sequence, and for the classifier output also the NCBI taxid(s), and genus-species of any classification(s).

These files are not really intended for human use, but are readable. Here we skip ten lines of sample metadata at the start, and all the sample-specific counts in columns 2 to 123, and the sequence in column 124, showing just the first and final two columns:

```
$ tail -n +10 summary/thapbi-pict.ITS1.onebp.tsv | head | cut -f 1,125,126
<SEE TABLE BELOW>
```

Viewing it like this is not ideal, although there are command line tools which help. You could instead open the file in R, Excel, etc:

#Marker/MD5_abundance	taxid	genus-species
ITS1/2e4f0ed53888ed39a2aee6d6d8e02	221518	Phytophthora pseudosyringae
ITS1/d9bc3879fdab3b4184c04bfbb5cf6	631361	Phytophthora austrocedri
ITS1/32159de6cbb6df37d084e31c37c30	67594	Phytophthora syringae
ITS1/ed15fefb7a3655147115fc28a8d6d6	78237	Phytophthora gonapodyides
ITS1/972db44c016a166de86a2bacab3f4	2056922	Phytophthora x cambivora
ITS1/c1a720b2005f101a9858107545726	78237	Phytophthora gonapodyides
ITS1/96e0e2f0475bd1617a4b05e778bb0	78237	Phytophthora gonapodyides
ITS1/f27df8e8755049e831b1ea4521ad6	2496075;2897317;2	Phytophthora aleatoria;Phytophthora alpina;Phytophthora cactorum
ITS1/21d6308d89d74b8ed493d73a2cb4	2056922	Phytophthora x cambivora

The first entry says the most abundance sequence with MD5 checksum `2e4f0ed53888ed39a2aee6d6d8e02206` was seen in a total of 163094 reads, and was classified as *Phytophthora pseudosyringae* (NCBI taxid 221518). Another common sequence has been matched to two closely related species *Phytophthora cambivora* (NCBI taxid 53983) and *Phytophthora x cambivora* (NCBI taxid 2056922).

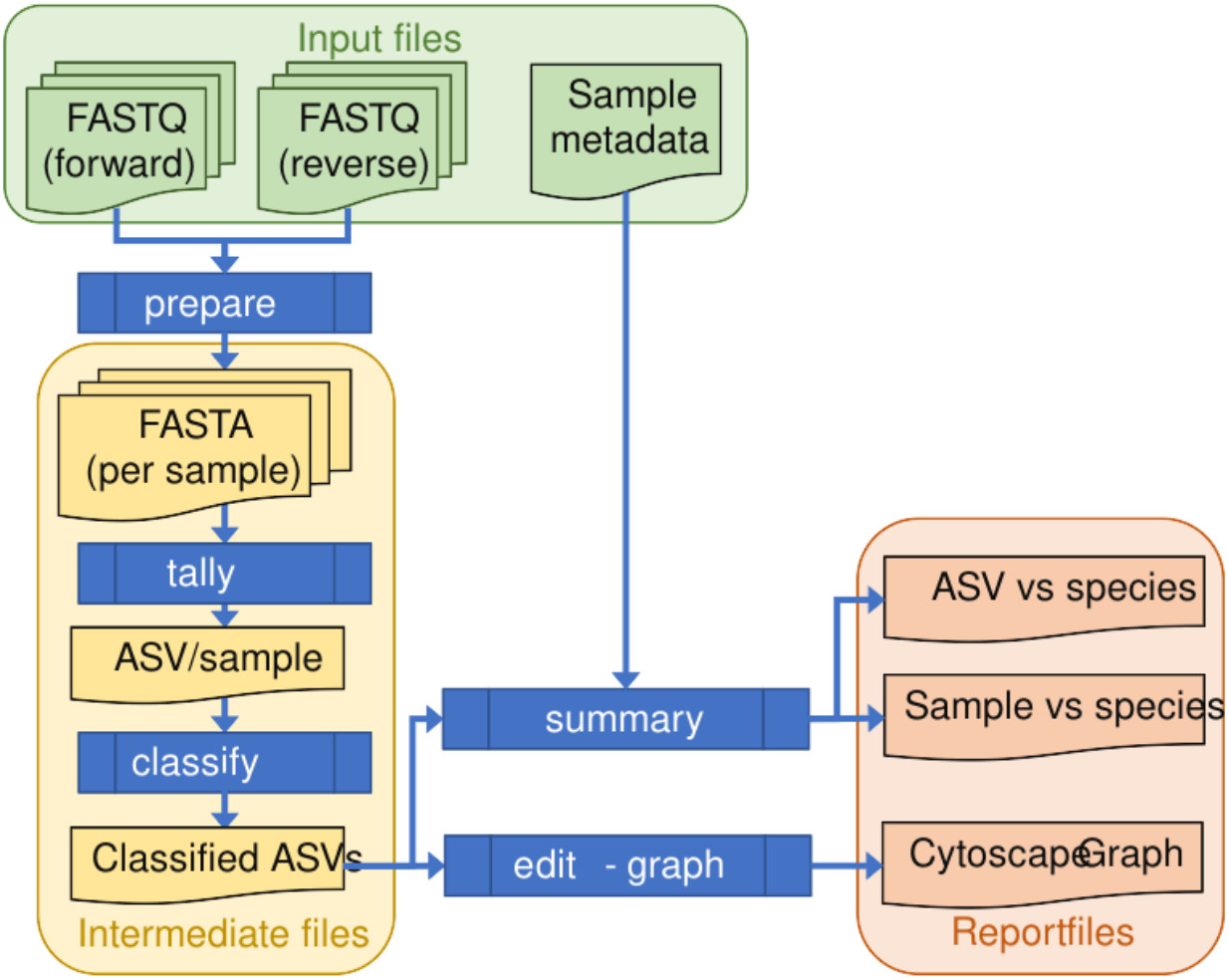
If you are familiar with the command line search tool `grep` and the regular expression syntax, you should find the format of these intermediate TSV files lends itself to some simple searches. For example, you could see which samples had matches to *Phytophthora rubi* using `grep` as follows:

```
$ grep "Phytophthora rubi" summary/thapbi-pict.ITS1.onebp.tsv | cut -f 1,125,126
ITS1/d8613e80b8803b13f7ea5d097f8fe46f_899  129364  Phytophthora rubi
$ grep d8613e80b8803b13f7ea5d097f8fe46f intermediate/ITS1/*.fasta
intermediate/ITS1/DNA10MIX_bycopynumber.fasta:>d8613e80b8803b13f7ea5d097f8fe46f_279
intermediate/ITS1/DNA10MIX_diluted25x.fasta:>d8613e80b8803b13f7ea5d097f8fe46f_349
intermediate/ITS1/DNA10MIX_undiluted.fasta:>d8613e80b8803b13f7ea5d097f8fe46f_271
```

The summary reports would also answer this particular question, but this kind of search can be useful for exploring specific questions.

4.1.4 Metadata

The *Quick Start* introduced the typical pipeline taking paired FASTQ files though to reports, and mentioned the idea of enhancing the reports with sample metadata.



In the following we will show the reports with and without metadata. As described earlier (see *Marker data*), `metadata.tsv` is a table of metadata based on table S1 in the paper, with 16 columns.

We will use `-c 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15` (or `--metacols`) meaning show columns 1 to 15 inclusive in the reports (in that order).

Finally, we will use `-x 16` or `--metaindex 16` to indicate column 16 contains cross references to the sequenced sample filename stems (semi-colon separated). They will be shown in this order.

This cross referencing idea is key to getting the best results from attaching metadata to your sequenced samples. Here is an abridged representation of the table, showing column one (site or control name), column two (altitude), and finally column 16 which has the filename stems of the sequence data belonging to this row of the table (semi-colon separated list).

#Site	Altitude	...	MiSeq Sample(s)
01	30	...	Site_1_sample_1; Site_1_sample_2; Site_1_sample_3; Site_1_sample_4; Site_1_sample_5; Site_1_sample_6; Site_1_sample_7; Site_1_sample_8; Site_1_sample_9-2; Site_1_sample_10
02	55	...	Site_2_sample_1; Site_2_sample_2; Site_2_sample_3; Site_2_sample_4; Site_2_sample_5; Site_2_sample_6; Site_2_sample_7; Site_2_sample_8; Site_2_sample_9; Site_2_sample_10
03	45	...	Site_3_sample_1; Site_3_sample_2; Site_3_sample_4; Site_3_sample_7; Site_3_sample_8; Site_3_sample_9
04	20	...	Site_4_sample_1; Site_4_sample_2; Site_4_sample_3; Site_4_sample_3-2; Site_4_sample_4; Site_4_sample_5; Site_4_sample_6; Site_4_sample_8; Site_4_sample_9; Site_4_sample_10
05	100	...	Site_5_sample_1; Site_5_sample_2; Site_5_sample_4; Site_5_sample_5; Site_5_sample_6; Site_5_sample_8; Site_5_sample_9
06	5	...	Site_6_sample_1; Site_6_sample_2-2; Site_6_sample_3-1; Site_6_sample_4; Site_6_sample_5-3; Site_6_sample_6; Site_6_sample_7-1; Site_6_sample_8-2; Site_6_sample_9; Site_6_sample_10
07	105	...	Site_7_sample_1; Site_7_sample_2; Site_7_sample_3; Site_7_sample_5; Site_7_sample_6; Site_7_sample_7; Site_7_sample_8; Site_7_sample_9; Site_7_sample_10
08	45	...	Site_8_sample_1; Site_8_sample_2; Site_8_sample_3; Site_8_sample_4; Site_8_sample_5-2; Site_8_sample_6; Site_8_sample_7; Site_8_sample_7-2; Site_8_sample_8; Site_8_sample_9
09	15	...	Site_9_sample_1; Site_9_sample_4-3; Site_9_sample_6; Site_9_sample_7; Site_9_sample_8; Site_9_sample_9; Site_9_sample_10
10	30	...	Site_10_sample_7; Site_10_sample_8
11	80	...	Site_11_sample_1; Site_11_sample_2; Site_11_sample_3; Site_11_sample_4; Site_11_sample_5; Site_11_sample_6; Site_11_sample_7; Site_11_sample_8; Site_11_sample_9; Site_11_sample_10
12	30	...	Site_12_sample_1; Site_12_sample_2; Site_12_sample_3-3; Site_12_sample_4; Site_12_sample_5-3; Site_12_sample_6; Site_12_sample_8; Site_12_sample_9; Site_12_sample_10
13	300	...	Site_13_sample_1; Site_13_sample_2; Site_13_sample_4; Site_13_sample_5; Site_13_sample_6; Site_13_sample_7; Site_13_sample_8; Site_13_sample_9; Site_13_sample_10
14	30	...	Site_14_sample_1-2; Site_14_sample_2; Site_14_sample_3; Site_14_sample_4; Site_14_sample_5; Site_14_sample_6; Site_14_sample_10
DNA1(		...	DNA10MIX_undiluted; DNA10MIX_diluted25x; DNA10MIX_bycopynumber
DNA16		...	DNA16MIX
NEGATIVE		...	NEGATIVE_firstplate; NEGATIVE_secondplate

Also note that in column one we have listed the numerical site names with leading zeros giving `01` to `14` to ensure they sort as expected.

4.1.5 Summary reports

Running thapbi_pict summary

The reports from the pipeline can be generated separately by the `thapbi_pict summary` command:

```
$ thapbi_pict summary -h
...
```

To mimic what the pipeline command would do, run the following:

```
$ thapbi_pict summary -i intermediate/ \
  summary/thapbi-pict.ITS1.onebp.tsv \
  -o summary/thapbi-pict.ITS1
...
```

Note the trailing slash \ at the end of the first line indicates the command continues on the next line. You can actually type this at the standard Linux command prompt (or include it in a copy and paste), or just enter this as one very long command.

We will look at the output in a moment, along side the equivalent reports generated with *metadata* (see linked discussion about column numbers):

```
$ thapbi_pict summary -i intermediate/ \
  summary/thapbi-pict.ITS1.onebp.tsv \
  -o summary/with-metadata.ITS1 \
  -t metadata.tsv -c 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -x 16
...
```

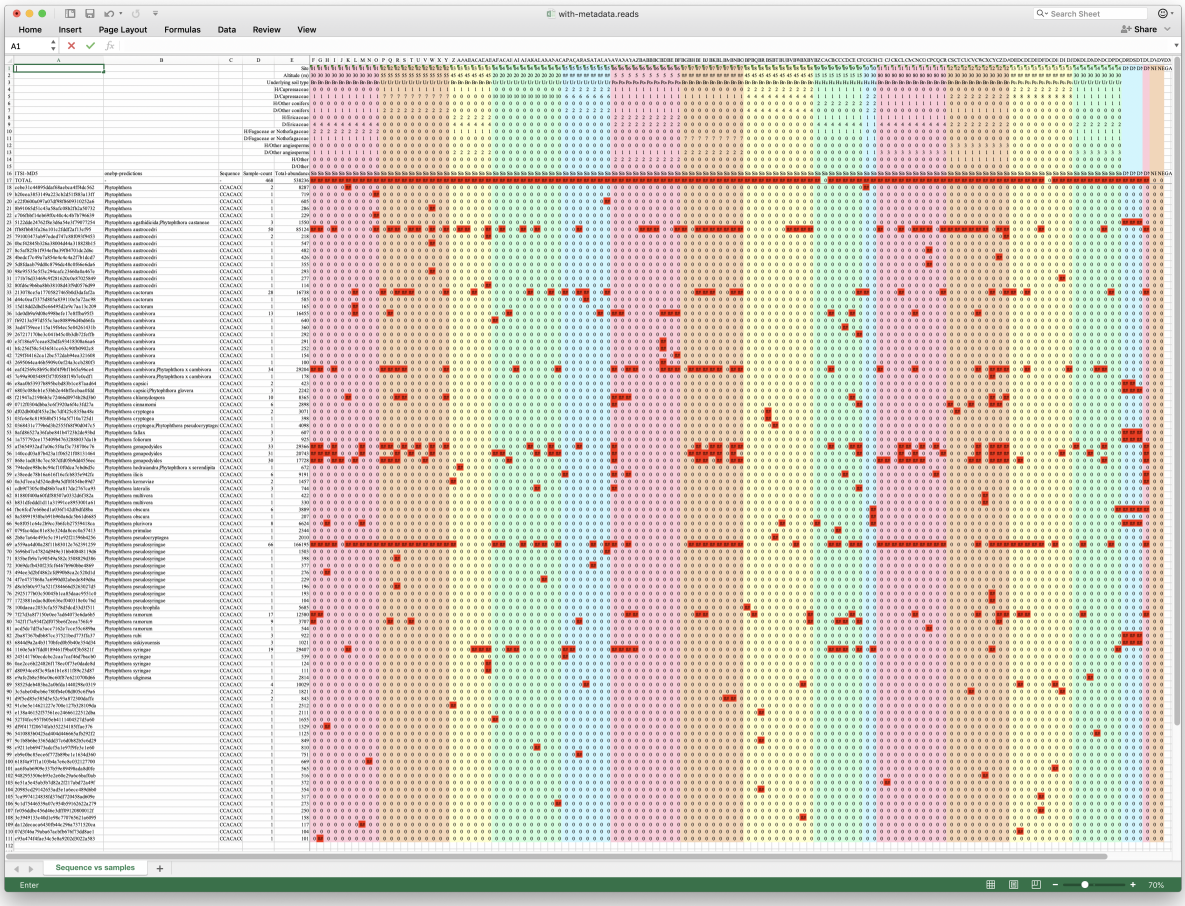
Both the read report and sample report are tables, produced as both computer-friendly plain text tab-separated variable (TSV), and human-friendly Excel (with colors and conditional formatting).

Read Report

The heart of the read report is a large table, of unique sequences (ASVs rows) versus sequenced samples (columns), with read abundance counts. There are additional columns with sequence information, and when *Metadata* is present, extra rows at the start with sample information.

This read report has a row for each unique sequence. The first columns are the marker name (here always "ITS1"), the unique sequence MD5 checksum, any species prediction, the sequence itself, the number of samples it was detected in above the threshold, the maximum number of reads with this sequence in any one sample, and the total number of reads (from samples where it was above the threshold). Then the main columns (one per sample) list the abundance of each unique sequence in that sample (if above the threshold).

In the Excel version, conditional formatting is used to highlight the non-zero counts with a red background. Furthermore, with metadata it will attempt to assign repeated bands of background color to groups (pink, orange, yellow, green, blue). In this example, each sample site gets a new color:



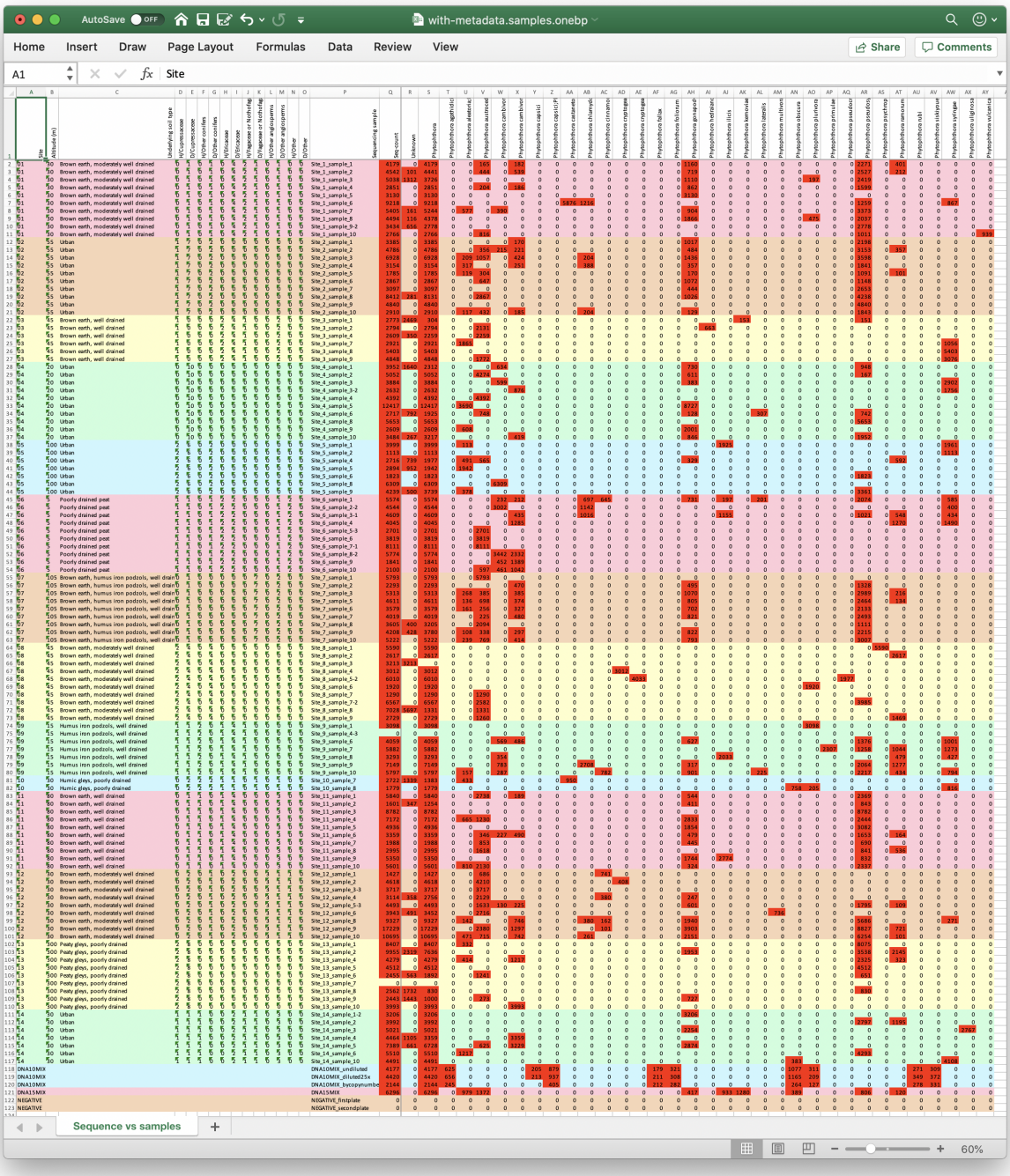
Typical sample naming schemes will result in replicates as neighbouring columns - meaning you should see very similar patterns of red (non-zero). Certainly in this dataset scanning horizontally we do see some sequences clearly show presence/absence patterns consistent with the samples.

The default row sorting will result in a dominant sequence being followed by any close variants assigned to the same species. Many of these rows will represent PCR artefacts found in just one or two samples. This contributes to the "halo" effect seen in the *Edit Graph* representation, discussed next.

Sample Report

The heart of the sample report is a table of samples (rows) versus species predictions (columns), with read abundance counts. There are additional columns with sample read counts, and when *Metadata* is present, extra columns at the start with sample information.

Here is a screenshot of the `summary/with-metadata.ITS1.samples.onebp.xlsx` file opened in Excel:



The metadata is in the first columns, then the sequence filename stem, a text summary of the species predictions, some inferred sequence count data, and the one column for each unique species or ambiguous species combinations.

Using the metadata each site has one or more rows in the same background color (pink, orange, yellow, green, blue, repeated), with one row for each time it was sequenced (the per-site sampling).

The values are total read counts for that row/column, with conditional formatting applied so non-zero entries have a bright red background.

For example, the final rows are the two DNA mixture controls (blue and pink) and the negative controls (orange). These

have almost no metadata, and the negative controls read counts are all zero.

The plain text table `with-metadata.ITS1.samples.onebp.xlsx` is the same, but without the colors and formatting. The files generated without metadata (`thapbi-pict.ITS1.samples.onebp.xlsx` etc) lack the extra columns and the background color bands.

The files without metadata start with the FASTQ filename stem as the inferred sample name in column 1:

```
$ cut -f 1 summary/thapbi-pict.ITS1.samples.onebp.tsv | head
#Sequencing sample
DNA10MIX_bycopynumber
DNA10MIX_diluted25x
DNA10MIX_undiluted
DNA15MIX
NEGATIVE_firstplate
NEGATIVE_secondplate
Site_10_sample_7
Site_10_sample_8
Site_11_sample_1
```

In contrast, the 15 extra metadata columns are inserted before this, and are used to sort the samples:

```
$ cut -f 1,16 summary/with-metadata.ITS1.samples.onebp.tsv | head
#Site  Sequencing sample
01     Site_1_sample_1
01     Site_1_sample_2
01     Site_1_sample_3
01     Site_1_sample_4
01     Site_1_sample_5
01     Site_1_sample_6
01     Site_1_sample_7
01     Site_1_sample_8
01     Site_1_sample_9-2
```

Like the FASTQ filename stems, the metadata is still sorted as strings, but by using leading zeros and `YYYY-MM-DD` style for any dates, you can achieve a logical presentation.

After the sequencing sample name (the FASTQ filename stem), we have the classification summary as a comma separated list - attempting to summarise the later per-species columns. Species listed here with (*) are where sequences matched multiple species equally well. For example, *Phytophthora andina*, *P. infestans*, and *P. ipomoeae*, share an identical ITS1 marker.

The next columns are derived from the data itself, reads counts in the samples as raw FASTQ, after read merging with Flash, primer trimming with Cutadapt, information about the abundance thresholds used (omitted below), the maximum ASV read count for non-spike-in or spike-in sequences, number of singletons, total number of reads for the accepted ASVs (i.e. passing the abundance threshold), and the number of unique ASVs accepted. It may be easier to look at this in Excel, but at the command line:

```
$ cut -f 16,18-20,24-28 summary/with-metadata.ITS1.samples.onebp.tsv | head
<SEE TABLE BELOW>
```

As a table:

Sequencing sample	Raw FASTQ	Flash	Cutadapt	Max non-spike	Max spike-in	Singletons	Accepted	Unique
Site_1_sample_1	6136	5900	5886	2269	0	692	4180	8
Site_1_sample_2	6135	5955	5947	2532	0	671	4548	8
Site_1_sample_3	6778	6484	6470	2146	0	579	5060	5
Site_1_sample_4	4145	3984	3974	1499	0	469	2852	7
Site_1_sample_5	4722	4232	4213	3130	0	433	3130	1
Site_1_sample_6	12633	12070	12034	5864	0	1217	9208	4
Site_1_sample_7	7560	7170	7141	3372	0	741	5402	5
Site_1_sample_8	6324	5956	5942	2037	0	630	4524	5
Site_1_sample_9-2	4542	4335	4331	2780	0	385	3436	2

Finally, we get to the main part of the sample table, one column per classifier result, with the number of reads. Picking out some examples:

```
$ cut -f 16,31,41,64 summary/with-metadata.ITS1.samples.onebp.tsv | head
<SEE TABLE BELOW>
```

As a table:

Sequencing sample	Phytophthora austrocedri	Phytophthora gonapodyides	Unknown
Site_1_sample_1	165	1158	0
Site_1_sample_2	445	718	101
Site_1_sample_3	0	1110	1313
Site_1_sample_4	204	861	0
Site_1_sample_5	0	3130	0
Site_1_sample_6	0	0	0
Site_1_sample_7	0	902	161
Site_1_sample_8	0	1863	116
Site_1_sample_9-2	0	0	656

Generally we hope to see single species predictions for each ASV, however when there are conflicts such as equally good matches, or a reference sequence that is shared between species, both are reported. For example:

```
$ cut -f 16,35 summary/with-metadata.ITS1.samples.onebp.tsv | head
<SEE TABLE BELOW>
```

As a table:

Sequencing sample	Phytophthora chlamydospora;Phytophthora x stagnum
Site_1_sample_1	0
Site_1_sample_2	0
Site_1_sample_3	0
Site_1_sample_4	0
Site_1_sample_5	0
Site_1_sample_6	1217
Site_1_sample_7	0
Site_1_sample_8	0
Site_1_sample_9-2	0

In this example, `Site_1_sample_6` had sequences matching both *Phytophthora chlamydospora* and *Phytophthora x stagnum*. These species are listed with a `(*)` suffix in the earlier classification summary column:

```
$ grep Site_1_sample_6 summary/with-metadata.ITS1.samples.onebp.tsv | cut -f 16,17
Site_1_sample_6  Phytophthora castanetorum, Phytophthora chlamydospora(*), Phytophthora␣
↪pseudosyringae, Phytophthora syringae, Phytophthora x stagnum(*)
```

4.1.6 Edit Graph

Running thapbi_pict edit-graph

This is not run as part of the pipeline command, but must be run separately:

```
$ thapbi_pict edit-graph -h
...
```

This command does not use metadata, but can optionally use the intermediate TSV files. It requires the sample tally file:

```
$ thapbi_pict edit-graph -i summary/thapbi-pict.ITS1.tally.tsv \
    -o summary/thapbi-pict.edit-graph.onebp.xgmml
...
```

This will generate an XGMML (eXtensible Graph Markup and Modeling Language) file by default, but you can also request other formats including PDF (which requires additional dependencies including GraphViz):

```
$ thapbi_pict edit-graph -i summary/thapbi-pict.ITS1.tally.tsv \
      -o summary/thapbi-pict.edit-graph.onebp.pdf -f pdf
...
```

Nodes and edges

In this context, we are talking about a graph in the mathematical sense of nodes connected by edges. Our nodes are unique sequences (which we can again label by the MD5 checksum), and the edges are how similar two sequences are. Specially, we are using the Levenshtein edit distance. This means an edit distance of one could be a single base substitution, insertion or deletion.

The tool starts by compiling a list of all the unique sequences in your samples (i.e. all the rows in the `thapbi_pict read-summary` report), and optionally all the unique sequences in the database. It then computes the edit distance between them all (this can get slow).

We build the network graph by adding edges for edits of up to three base pairs (by default). This gives small connected components or sub-graphs which are roughly at the species level.

Redundant edges are dropped, for example if *A* is one edit away from *B*, and *B* is one edit away from *C*, there is need to draw the two edit line from *A* to *C*.

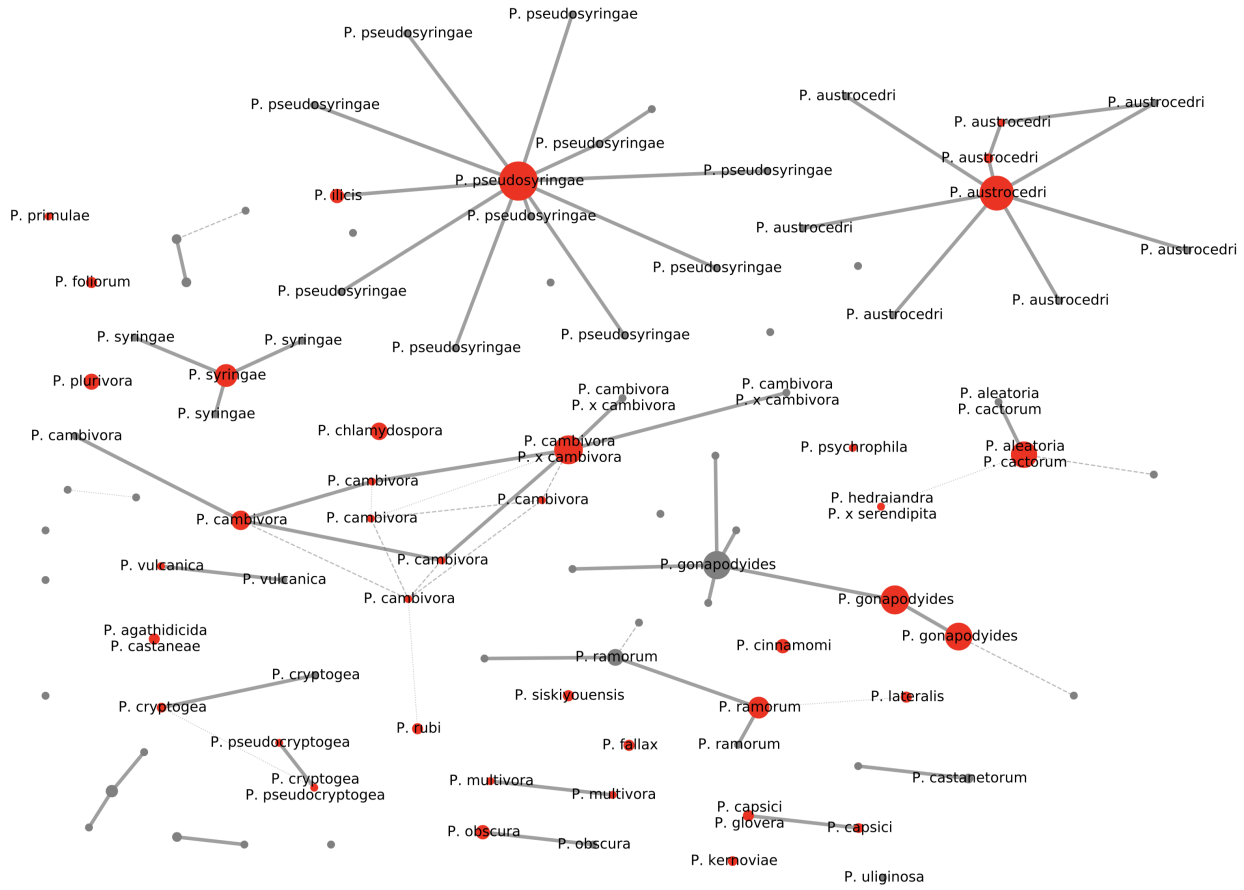
We draw the nodes as circles, scaled by the number of samples that unique sequence appeared in. If that exact sequence is in the database, is it colored according to genus, defaulting to grey.

Color	RGB value	Meaning
Red	FF0000	*Phytophthora*
Lime	00FF00	*Peronospora*
Blue	0000FF	*Hyaloperonospora*
Yellow	FFFF00	*Bremia*
Cyan	00FFFF	*Pseudoperonospora*
Magenta	FF00FF	*Plasmopara*
Maroon	800000	*Nothophytophthora*
Olive	808000	*Peronosclerospora*
Green	008000	*Perofascia*
Purple	800080	*Paraperonospora*
Teal	008080	*Protobremia*
Dark red	8B0000	Other known genus
Dark orange	FF8C00	Conflicting genus
Orange	FFA500	Synthetic sequence
Grey	808080	Not in the database

The edges are all grey, solid for a one base pair edit distance, dashed for a two base pair edit distance, and dotted for a three base pair edit distance.

Viewing the PDF

You should be able to open the PDF file easily, and get something like this - lots of red circles for *Phytophthora*, some grey circles for sequences not in the database, and plenty of grey straight line edges between them.



In the PDF (and XGMML) output, nodes are coloured by genus (red for *Phytophthora*), but only labelled if in the database at species level.

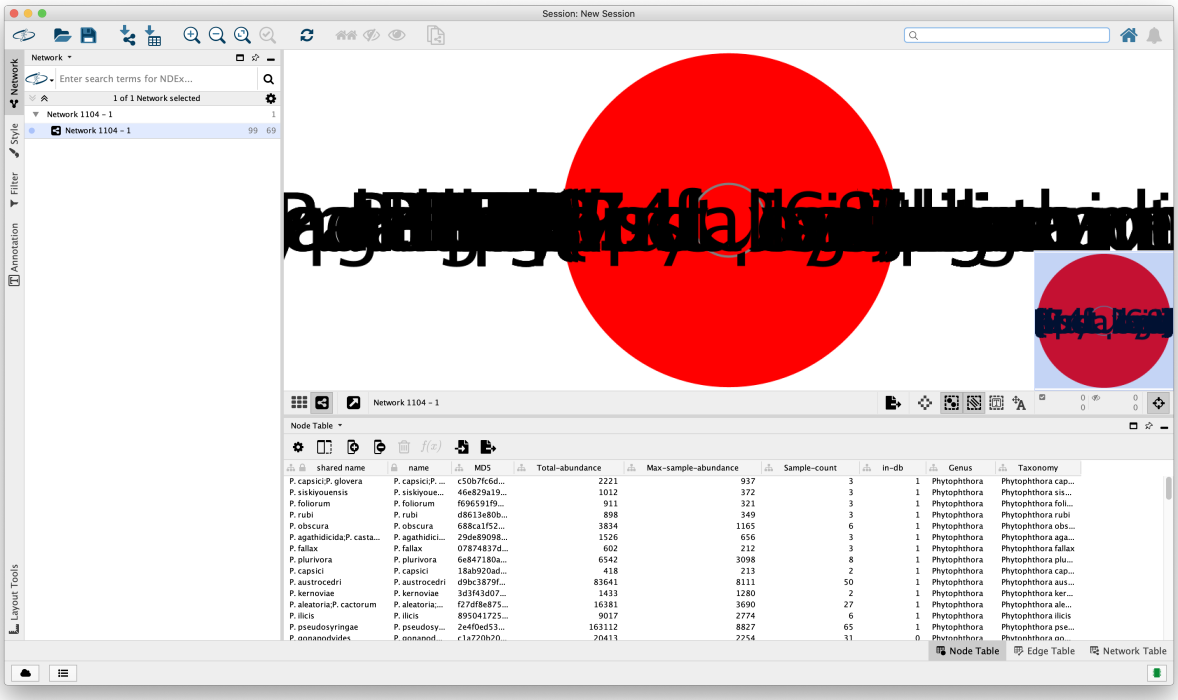
The edges are solid for a one base pair edit distance, dashed for a two base pair edit distance, and dotted for a three base pair edit distance. All grey.

Viewing the XGMML

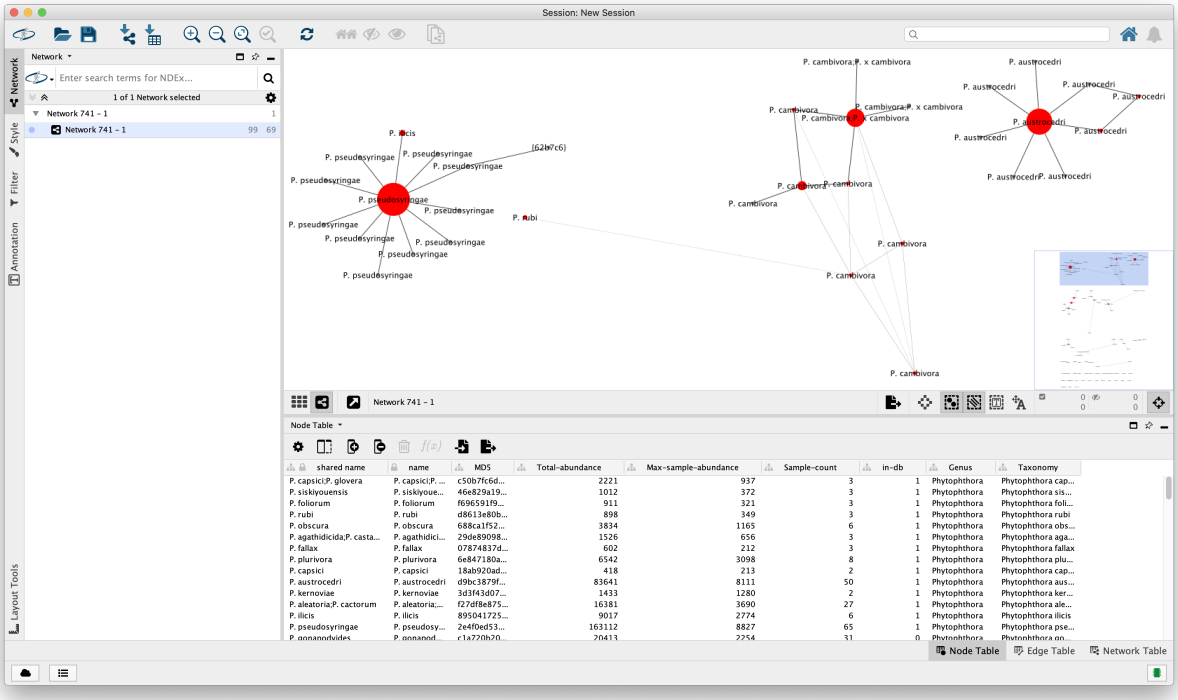
You should be able to open the PDF file easily, and while it is interesting it is read only and non-interactive. This is where the XGMML output shines. You will need to install the free open source tool Cytoscape to use this.

Open Cytoscape, and from the top level menu select `File`, `Import`, `Network from file...`, then select `summary/thapbi-pict.edit-graph.onebp.xgmml` (the XGMML file created above).

You should get something like this, where initially all the nodes are drawn on top of each other:



From the top level menu select “Layout”, “Perfuse Force Directed Layout”, “Edit-distance-weight”, and you should then see something prettier - if you zoom in you should see something like this:



This time you can interact with the graph, moving nodes about with the mouse, try different layouts, view and search the attributes of the nodes and edges.

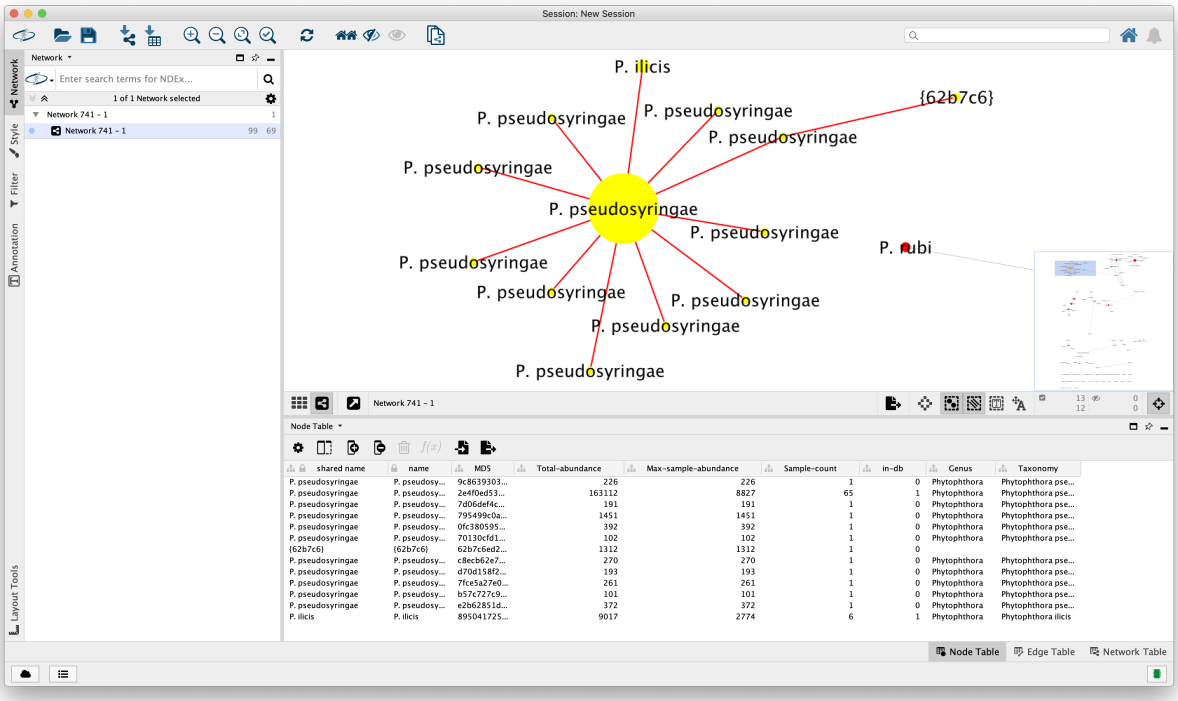
Here the nodes are labelled with the species if they were in the database at species level, or otherwise as the start of the MD5 checksum in curly brackets (so that they sort nicely). The default node colors are as in the PDF output, likewise the grey edge styles.

The node attributes include the full MD5 (so you can lookup the full sequence or classification results for any node of interest), sample count, total read abundance (both numbers shown in the `thapbi_pict summary` reports), genus (allowing you to do your own color scheme), and species if known.

The edge attributes include `Edit-distance` (values 1, 2, 3 for number of base pairs difference between sequences) and matching `Edit-distance-weight` (values 3, 2, 1 used earlier for the layout where we prioritise the small edit distance edges).

Halo effect

In this final screenshot we have zoomed in and selected all 11 nodes in the connected component centered on *P. pseudosyringae* (Cytoscape highlights selected nodes in yellow):



The node table view is automatically filtered to show just these nodes, and we can see that all the grey nodes appeared in only one sample each - with the *P. pseudosyringae* entry in the database in 66 samples, while the one base away *P. ilics* sequence was in 6 samples.

This kind of grey-node halo around highly abundance sequences is more common when plotting larger datasets. It is consistent with PCR artefacts occurring in just one (or two) samples giving rise to (almost) unique sequences based on the template sequence.

4.1.7 Pipeline with metadata

Running thapbi-pict pipeline

Having run all the steps of the typical pipeline individually, we now return to the top level `thapbi_pict pipeline` command:

```
$ thapbi_pict pipeline -h
...
```

Assuming you have the FASTQ files in `raw_data/`, we can run the pipeline command as follows, and should get multiple output report files:

```
$ thapbi_pict pipeline -i raw_data/ -s intermediate/ \
  -o summary/thapbi-pict
...
$ ls -1 summary/thapbi-pict.*
summary/thapbi-pict.ITS1.onebp.tsv
summary/thapbi-pict.ITS1.reads.onebp.tsv
summary/thapbi-pict.ITS1.reads.onebp.xlsx
summary/thapbi-pict.ITS1.samples.onebp.tsv
summary/thapbi-pict.ITS1.samples.onebp.xlsx
summary/thapbi-pict.ITS1.tally.tsv
summary/thapbi-pict.edit-graph.onebp.pdf
summary/thapbi-pict.edit-graph.onebp.xgmml
```

As described for the *prepare-reads step* we should also specify which of the samples are negative controls, which may be used to increase the plate level minimum abundance threshold:

```
$ thapbi_pict pipeline -i raw_data/ -s intermediate/ \
  -o summary/thapbi-pict -n raw_data/NEGATIVE*.fastq.gz
...
```

And, as described for the *summary reports*, we can provide metadata:

```
$ thapbi_pict pipeline -i raw_data/ -s intermediate/ \
  -o summary/with-metadata -n raw_data/NEGATIVE*.fastq.gz \
  -t metadata.tsv -c 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -x 16
...
```

Finally, as we will review next, we can ask the pipeline to assess the results against any expected sample species classifications:

```
$ thapbi_pict pipeline -i raw_data/ expected/ -s intermediate/ \
  -o summary/with-metadata -n raw_data/NEGATIVE*.fastq.gz \
  -t metadata.tsv -c 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -x 16
...
$ ls -1 summary/with-metadata.*
summary/with-metadata.ITS1.onebp.tsv
summary/with-metadata.ITS1.assess.confusion.onebp.tsv
summary/with-metadata.ITS1.assess.onebp.tsv
summary/with-metadata.ITS1.assess.tally.onebp.tsv
summary/with-metadata.ITS1.reads.onebp.tsv
summary/with-metadata.ITS1.reads.onebp.xlsx
```

(continues on next page)

(continued from previous page)

```
summary/with-metadata.ITS1.samples.onebp.tsv
summary/with-metadata.ITS1.samples.onebp.xlsx
summary/with-metadata.ITS1.tally.tsv
```

Here we also used `-o` (or `--output`) to specify a different stem for the report filenames.

Conclusions

For the THAPBI Phyto-Threats project our datasets span multiple plates, but we want to set plate-specific minimum abundance thresholds. That is taken care of as long as each plate is in its own directory. For example, you might have `raw_data/plate_NNN/*.fastq.gz` and run the pipeline with `-i raw_data/`).

However, while you could run the pipeline command on all the data in one go, with access to a computer cluster it will likely be faster to run at least the (slowest) `prepare-reads` stage on separate cluster nodes (e.g. one cluster job for each plate).

4.1.8 Assessing the classifier

This sample dataset includes two positive control mock communities. We know the species which went into the two different DNA mixes used, so for each sequenced positive control sample we can compare the expected list of species with the predicted list of species, and thus count true positives, false positives, false negatives, etc.

We will first do this by hand, and then explore the tool's own built in assessment framework.

Counting species for one sample by hand

The woody hosts dataset had two positive control mixes. From the first plate, a set of 15 *Phytophthora* species (listed here alphabetically):

- *Phytophthora austrocedri*
- *Phytophthora boehmeriae*
- *Phytophthora cactorum*
- *Phytophthora cambivora* (now *Phytophthora x cambivora*)
- *Phytophthora chlamydospora*
- *Phytophthora cinnamomi*
- *Phytophthora gonapodyides*
- *Phytophthora ilicis*
- *Phytophthora kernoviae*
- *Phytophthora lateralis*
- *Phytophthora obscura*
- *Phytophthora plurivora*
- *Phytophthora pseudosyringae*
- *Phytophthora ramorum*
- *Phytophthora syringae*

Quoting from the *sample summary report*, using the default settings for classification of `DNA15MIX`, we got:

```
$ grep DNA15MIX summary/thapbi-pict.ITS1.samples.onebp.tsv | cut -f 2
Phytophthora aleatoria(*), Phytophthora alpina(*), Phytophthora austrocedri,␣
↪Phytophthora cactorum(*), Phytophthora gonapodyides, Phytophthora ilicis, Phytophthora␣
↪kernoviae, Phytophthora obscura, Phytophthora pseudosyringae, Phytophthora ramorum
```

Or, as a list:

- *Phytophthora aleatoria* (uncertain/ambiguous)
- *Phytophthora alpina* (uncertain/ambiguous)
- *Phytophthora austrocedri*
- *Phytophthora cactorum* (uncertain/ambiguous)
- *Phytophthora gonapodyides*
- *Phytophthora ilicis*
- *Phytophthora kernoviae*
- *Phytophthora obscura*
- *Phytophthora pseudosyringae*
- *Phytophthora ramorum*

The good news is that eight are correct classifications (eight true positives, 8 TP), but two false positives (2 FP). Those false positives *Phytophthora alpina* and *P. aleatoria* are indistinguishable from *P. cactorum*, a problem flagged via the `conflicts` command:

```
$ thapbi_pict conflicts | grep cactorum
f27df8e8755049e831b1ea4521ad6eb3  species  Phytophthora aleatoria;Phytophthora alpina;
↪Phytophthora cactorum
$ grep f27df8e8755049e831b1ea4521ad6eb3 intermediate/ITS1/DNA15MIX.fasta
>f27df8e8755049e831b1ea4521ad6eb3_981
```

The bad news is we are missing seven expected species (seven false negatives, 7 FN):

- *Phytophthora boehmeriae*
- *Phytophthora chlamydospora*
- *Phytophthora cinnamomi*
- *Phytophthora lateralis*
- *Phytophthora plurivora*
- *Phytophthora syringae*
- *Phytophthora x cambivora*

We will return to interpretation after showing how to get the tool to compute these FP, FP and FN values.

The positive controls from the second plate had a different mix of ten *Phytophthora* species, again listed alphabetically:

- *Phytophthora boehmeriae*
- *Phytophthora cactorum*
- *Phytophthora capsici*
- *Phytophthora castaneae*

- *Phytophthora fallax*
- *Phytophthora foliorum*
- *Phytophthora obscura*
- *Phytophthora plurivora*
- *Phytophthora rubi*
- *Phytophthora siskiyouensis*

Again referring to the sample summary report from running with default settings, for `DNA10MIX_undiluted` and `DNA10MIX_diluted25x` we got:

- *Phytophthora agathidicida* (uncertain/ambiguous)
- *Phytophthora capsici*
- *Phytophthora castaneae* (uncertain/ambiguous)
- *Phytophthora fallax*
- *Phytophthora foliorum*
- *Phytophthora gloveri* (uncertain/ambiguous)
- *Phytophthora obscura*
- *Phytophthora plurivora*
- *Phytophthora rubi*
- *Phytophthora siskiyouensis*

Plus the results from `DNA10MIX_bycopynumber` were almost the same - but this time there wasn't a sequence only matched to *P. capsici*, so that was also flagged as "(uncertain/ambiguous)".

Leaving aside the ambiguous qualifier, there are ten species predictions, but only nine are correct (9 TP: *P. capsici*, *P. castaneae*, *P. fallax*, *P. foliorum*, *P. obscura*, *P. plurivora*, *P. rubi*, *P. siskiyouensis*), with two wrong guesses (2 FP: *P. agathidicida* and *P. gloveri*), and two missing predictions (2 FN: *P. boehmeriae* and *P. cactorum*).

The uncertain/ambiguous prediction of *Phytophthora agathidicida* is easily explained, it comes from a sequence present in all three samples with MD5 checksum `5122dde24762f8e3d6a54e3f79077254`, and this exact sequence is in the database with entries for both *Phytophthora castaneae* (which was in the DNA control mixture) and also *Phytophthora agathidicida* (e.g. accession KP295308).

You can confirm this by looking at the sample tally TSV files, e.g. using grep to find the unique sequence matched to this species, and the sample counts for that sequence:

```
$ grep "Phytophthora agathidicida" summary/thapbi-pict.ITS1.onebp.tsv | cut -f 1,125,126
ITS1/29de890989becddc5e0b10ecbbc11b1a_1524  1642459;1642465  Phytophthora agathidicida;
↪Phytophthora castaneae
$ grep -E "(Sequence|29de890989becddc5e0b10ecbbc11b1a)" \
  summary/thapbi-pict.ITS1.tally.tsv | cut -f 2-5
DNA10MIX_bycopynumber  DNA10MIX_diluted25x  DNA10MIX_undiluted  DNA15MIX
245                    655                  624                 0
$ thapbi_pict conflicts | grep 29de890989becddc5e0b10ecbbc11b1a
29de890989becddc5e0b10ecbbc11b1a  species  Phytophthora agathidicida;Phytophthora␣
↪castaneae
```

The same applies to *Phytophthora capsici* and *Phytophthora gloveri*. i.e. These false positives are unavoidable.

As noted above, the woody hosts paper concluded the failure to detect *P. boehmeriae* in either DNA mix was due to inefficient primer annealing in a species mixture. We have an unexpected FN for *P. cactorum* though.

Running thapbi_pict assess for one sample

Comparing a few samples like this by hand is one thing, but doing it at scale requires automation. For assessing changes to the classifier method and database, we mainly run `thapbi_pict assess` against a set of single isolate positive controls. This requires a computer readable files listing the expected species in a particular format.

```
$ thapbi_pict assess -h
...
```

The "known" file uses the same column based layout as the intermediate TSV files, but while you can provide the expected species for each unique sequence in the sample, this can be simplified to a single wildcard * line followed by all the NCBI taxids (optional), and species names using semi-colon separators.

The simplest way to run the assess command is to tell it two TSV input filenames, named `<sample_name>.known.tsv` (the expected results) and `<sample_name>.<method>.tsv` (from running *thapbi_pict classify* ` on `<sample_name>.fasta`). However, although early versions of the pipeline did this, it has for a long time combined the samples for classification - partly for speed.

Instead we typically pass the assess command the sample-tally TSV file listing how many of each unique sequence were found in each sample, the classifier TSV listing the species assigned to each sequence, and one or more per-sample `<sample_name>.known.tsv` expected results files.

The assess command will default to printing its tabular output to screen - shown here abridged after piping through the `cut` command to pull out just the first five columns from the 15 species mix:

```
$ thapbi_pict assess -i summary/thapbi-pict.ITS1.onebp.tsv \
  expected/DNA15MIX.known.tsv | cut -f 1-5
Assessed onebp vs known in 2 files (260 species; 1 samples)
#Species                      TP  FP  FN  TN
OVERALL                       8   2   7   243
Phytophthora aleatoria        0   1   0   0
Phytophthora alpina           0   1   0   0
Phytophthora austrocedri      1   0   0   0
Phytophthora boehmeriae       0   0   1   0
Phytophthora cactorum         1   0   0   0
Phytophthora chlamydospora    0   0   1   0
Phytophthora cinnamomi        0   0   1   0
Phytophthora gonapodyides     1   0   0   0
Phytophthora ilicis           1   0   0   0
Phytophthora kernoviae        1   0   0   0
Phytophthora lateralis        0   0   1   0
Phytophthora obscura          1   0   0   0
Phytophthora plurivora        0   0   1   0
Phytophthora pseudosyringae   1   0   0   0
Phytophthora ramorum          1   0   0   0
Phytophthora syringae         0   0   1   0
Phytophthora x cambivora      0   0   1   0
OTHER 243 SPECIES IN DB       0   0   0   243
```

More usually, you would output to a named file, and look at that:

```
$ thapbi_pict assess -i summary/thapbi-pict.ITS1.onebp.tsv \
  expected/DNA15MIX.known.tsv -o DNA15MIX.assess.tsv
Assessed onebp vs known in 2 files (260 species; 1 samples)
$ cut -f 1-5,9,11 DNA15MIX.assess.tsv
<SEE TABLE BELOW>
```

You should be able to open this `DNA15MIX.assess.tsv` file in R, Excel, etc, and focus on the same column selection:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	8	2	7	243	0.64	0.529
Phytophthora aleatoria	0	1	0	0	0.00	1.000
Phytophthora alpina	0	1	0	0	0.00	1.000
Phytophthora austrocedri	1	0	0	0	1.00	0.000
Phytophthora boehmeriae	0	0	1	0	0.00	1.000
Phytophthora cactorum	1	0	0	0	1.00	0.000
Phytophthora chlamydospora	0	0	1	0	0.00	1.000
Phytophthora cinnamomi	0	0	1	0	0.00	1.000
Phytophthora gonapodyides	1	0	0	0	1.00	0.000
Phytophthora ilicis	1	0	0	0	1.00	0.000
Phytophthora kernoviae	1	0	0	0	1.00	0.000
Phytophthora lateralis	0	0	1	0	0.00	1.000
Phytophthora obscura	1	0	0	0	1.00	0.000
Phytophthora plurivora	0	0	1	0	0.00	1.000
Phytophthora pseudosyringae	1	0	0	0	1.00	0.000
Phytophthora ramorum	1	0	0	0	1.00	0.000
Phytophthora syringae	0	0	1	0	0.00	1.000
Phytophthora x cambivora	0	0	1	0	0.00	1.000
OTHER 243 SPECIES IN DB	0	0	0	243	0.00	0.000

The `OVERALL` line tells us that there were 8 true positives, 2 false positives, 7 false negatives, and 226 true negatives. The final number needs a little explanation. First, 8+2+7+226 = 243, which is the number of species in the database. With only one sample being considered, 226 is the number of other species in the database which the tool correctly says are not present.

Following this we get one line per species, considering this species in isolation (making this a traditional and simpler to interpret classification problem). Here there is only one sample, so this time TP+FP+FN+TN=1.

You can easily spot the 2 FP in this layout, *Phytophthora alpina* and *P. aleatoria*, or the 7 FN.

The additional columns (not all shown here) include traditional metrics like sensitivity, specificity, precision, F1, and Hamming loss. We've shown F1 or F-measure here (from zero to one for perfect recall), plus our own metric provisionally called *Ad hoc loss* which is a modification of the Hamming loss without using the true negative count (which we expect to always be very large as the database will contain many species, while a community might contain only ten).

Doing that for one of the 10 species mixtures:

```
$ thapbi_pict assess -i summary/thapbi-pict.ITS1.onebp.tsv \
  expected/DNA10MIX_undiluted.known.tsv -o DNA10MIX.assess.tsv
Assessed onebp vs known in 2 files (260 species; 1 samples)
$ cut -f 1-5,9,11 DNA10MIX.assess.tsv
<SEE TABLE BELOW>
```

As this is still only one sample, new table `DNA10MIX.assess.tsv` is very similar to what we had before:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	8	2	2	248	0.80	0.333
Phytophthora agathidicida	0	1	0	0	0.00	1.000
Phytophthora boehmeriae	0	0	1	0	0.00	1.000
Phytophthora cactorum	0	0	1	0	0.00	1.000
Phytophthora capsici	1	0	0	0	1.00	0.000
Phytophthora castaneae	1	0	0	0	1.00	0.000
Phytophthora fallax	1	0	0	0	1.00	0.000
Phytophthora foliorum	1	0	0	0	1.00	0.000
Phytophthora gloveri	0	1	0	0	0.00	1.000
Phytophthora obscura	1	0	0	0	1.00	0.000
Phytophthora plurivora	1	0	0	0	1.00	0.000
Phytophthora rubi	1	0	0	0	1.00	0.000
Phytophthora siskiyouensis	1	0	0	0	1.00	0.000
OTHER 248 SPECIES IN DB	0	0	0	248	0.00	0.000

It is clear from the metrics that the classifier is performing better on the second 10 species mock community.

Assessing multiple samples

Next, let's run the assess command on all four positive control samples, by giving the combined intermediate filenames, and *all* the expected files:

```
$ thapbi_pict assess -i summary/thapbi-pict.ITS1.onebp.tsv \
  expected/ -o thabpi-pict.ITS1.assess.tsv
Assessed onebp vs known in 5 files (260 species; 4 samples)
$ cut -f 1-5,9,11 thabpi-pict.ITS1.assess.tsv
<SEE TABLE BELOW>
```

New table `thabpi-pict.ITS1.assess.tsv` is similar, but notice all the per-species lines have TP+FP+FN+TN=4 as there were 4 samples:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	32	8	13	987	0.75	0.396
Phytophthora agathidicida	0	3	0	1	0.00	1.000
Phytophthora aleatoria	0	1	0	3	0.00	1.000
Phytophthora alpina	0	1	0	3	0.00	1.000
Phytophthora austrocedri	1	0	0	3	1.00	0.000
Phytophthora boehmeriae	0	0	4	0	0.00	1.000
Phytophthora cactorum	1	0	3	0	0.40	0.750
Phytophthora capsici	3	0	0	1	1.00	0.000
Phytophthora castaneae	3	0	0	1	1.00	0.000
Phytophthora chlamydospora	0	0	1	3	0.00	1.000
Phytophthora cinnamomi	0	0	1	3	0.00	1.000
Phytophthora fallax	3	0	0	1	1.00	0.000
Phytophthora foliorum	3	0	0	1	1.00	0.000
Phytophthora gloveri	0	3	0	1	0.00	1.000
Phytophthora gonapodyides	1	0	0	3	1.00	0.000
Phytophthora ilicis	1	0	0	3	1.00	0.000
Phytophthora kernoviae	1	0	0	3	1.00	0.000
Phytophthora lateralis	0	0	1	3	0.00	1.000
Phytophthora obscura	4	0	0	0	1.00	0.000
Phytophthora plurivora	3	0	1	0	0.86	0.250
Phytophthora pseudosyringae	1	0	0	3	1.00	0.000
Phytophthora ramorum	1	0	0	3	1.00	0.000
Phytophthora rubi	3	0	0	1	1.00	0.000
Phytophthora siskiyouensis	3	0	0	1	1.00	0.000
Phytophthora syringae	0	0	1	3	0.00	1.000
Phytophthora x cambivora	0	0	1	3	0.00	1.000
OTHER 235 SPECIES IN DB	0	0	0	940	0.00	0.000

This time the `OVERALL` line says we had 32 TP, 8 FP, 13 FN and 827 TN. Their total, 32+8+13+927 = 980 = 4 * 245, is the number of samples times the number of species in the database.

Running assessment as part of pipeline

Provided they follow the expected naming convention, if you include your control files `*.known.tsv` as one of the pipeline inputs, it will call the classifier assessment after running the classifier and producing the main reports:

```
$ thapbi_pict pipeline -i raw_data/ expected/ -s intermediate/ \
  -o summary/with-metadata -n raw_data/NEGATIVE*.fastq.gz \
  -t metadata.tsv -c 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -x 16
...
$ ls -1 summary/with-metadata.*
summary/with-metadata.ITS1.assess.confusion.onebp.tsv
summary/with-metadata.ITS1.assess.onebp.tsv
summary/with-metadata.ITS1.assess.tally.onebp.tsv
summary/with-metadata.ITS1.onebp.tsv
summary/with-metadata.ITS1.reads.onebp.tsv
summary/with-metadata.ITS1.reads.onebp.xlsx
summary/with-metadata.ITS1.samples.onebp.tsv
summary/with-metadata.ITS1.samples.onebp.xlsx
summary/with-metadata.ITS1.tally.tsv
```

(continues on next page)

(continued from previous page)

```
$ diff summary/with-metadata.ITS1.assess.onebp.tsv \
  thabpi-pict.ITS1.assess.tsv
```

Output file `summary/with-metadata.ITS1.assess.onebp.tsv` will match the output above.

Interpretation of the mock communities

Running our pipeline with the default settings results in a number of false positives (all unavoidable as they come from conflicting marker sequences in the database, see the `thapbi_pict conflicts` command), and some false negatives (on top of the explained absence of *Phytophthora boehmeriae*). Specifically we have 6 unexplained false negatives on the 15 species mix, and are missing *Phytophthora cactorum* in all three samples of the 10 species mix.

This means that with the default settings THAPBI PICT gives a more cautious set of predictions than the `metapy` tool used in the original data analysis (see Riddell et al. (2019) Table 1, Table 2) which appears to consider even singletons.

Attempting to compare the results in their Table 1 with our own numbers is complicated since it appears to show just one of the 10 species mixes (so the TP count is out of 10) while we used all three (for a TP count out of 30).

We can therefore pick a single representative sample for the 10 species mix, to make direct comparison more straight forward:

```
$ thapbi_pict assess -i summary/thapbi-pict.ITS1.onebp.tsv \
  expected/DNA15MIX.known.tsv expected/DNA10MIX_undiluted.known.tsv \
  | head -n 2 | cut -f 1-5,9,11
Assessed onebp vs known in 3 files (260 species; 2 samples)
#Species  TP  FP  FN  TN   F1    Ad-hoc-loss
OVERALL   16  4   9   491  0.71  0.448
```

We can recover most of the missing species (the FN) by dropping the minimum abundance thresholds (which requires deleting the intermediate FASTA files, or using a different intermediate folder, and re-running with lower settings for `-a` and `-f`), at the cost of more FP.

For instance, we find traces of *P. syringae* with less than 10 reads in the 15 species mix (consistent with Table 2), and even *P. boehmeriae* with less than 10 reads in two of the 10 species mix (not reported in Table 2).

Interestingly even excluding only singletons (using `-a 2 -f 0`), we didn't find any matches to *Phytophthora cactorum* in the three samples of the 10 species mix. However, there is a sequence perfectly matching database entries for *P. idaei* present at around 40 to 60 copies, and in light of the original paper, this is likely what was intended to be in the mixture as *P. cactorum*.

Again even excluding only singletons, we didn't find any matches to *P. plurivora* in the 15 species mix (Table 2 in the original paper suggests present with only 2 reads).

We can optimise the threshold by maximising the F1 score and minimising ad-hoc-loss for these two samples. This is done at the end of the `run.sh` script with a simple parameter sweep of the absolute threshold (`-a`) with the fractional threshold unused (`-f 0`). This produces a simple table:

```
$ cut -f 1-5,9,11 summary/mocks_a2.assess-vs-abundance.tsv
<SEE TABLE BELOW>
```

Open the table in Excel if you prefer, the columns of particular interest:

#Threshold	TP	FP	FN	TN	F1	Ad-hoc-loss
A=2	22	17	3	478	0.69	0.476
A=10	20	9	5	486	0.74	0.412
A=20	20	8	5	487	0.75	0.394
A=30	19	8	6	487	0.73	0.424
A=40	19	6	6	489	0.76	0.387
A=50	19	5	6	490	0.78	0.367
A=60	18	5	7	490	0.75	0.400
A=70	18	5	7	490	0.75	0.400
A=80	18	5	7	490	0.75	0.400
A=90	16	4	9	491	0.71	0.448
A=100	16	4	9	491	0.71	0.448

This suggests the optimal absolute abundance threshold for these two samples is in the region of 50 reads, giving 19 TP, 5 FP, and 6 FN for an F1 of 0.78 and ad-hoc-loss of 0.367. If we run the optimisation on all four samples (one with 15 species, three with 10 species), this suggests somewhere in between this and the default of 100.

4.2 Environmental Oomycetes ITS1

The first *worked example* looked at *Phytophthora* ITS1 data from woody-host trees, using the same PCR primers as the THAPBI PICT defaults, and the default database of *Phytophthora* ITS1 data provided.

Here we re-analyse a published dataset from a different group, doing Illumina MiSeq ITS1 amplicon sequencing of irrigation water samples from Oregon:

> Redekar *et al.* (2019) Diversity of *Phytophthora*, *Pythium*, and *Phytopythium* species in recycled irrigation water in a container nursery. https://doi.org/10.1094/PBIOMES-10-18-0043-R

Different PCR primers were used to cover *Pythium* and *Phytopythium* as well as *Phytophthora*. This requires a new database of markers.

4.2.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/recycled_water/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

The documentation goes through running each step of the analysis gradually, including building a custom database, before finally calling pipeline command to do it all together. We provide script `run.sh` to do the final run-though automatically, but encourage you to follow along the individual steps first.

FASTQ data

File `PRJNA417859.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata. With a little scripting to extract the relevant *sample metadata* for use with THAPBI PICT this was reformatted as `metadata.tsv` (see below).

Script `setup.sh` will download the raw FASTQ files for Redekar *et al.* (2019) from https://www.ebi.ac.uk/ena/data/view/PRJNA417859 - you could also use https://www.ncbi.nlm.nih.gov/bioproject/PRJNA417859/

It will download 768 raw FASTQ files (384 pairs), taking about 4.8GB on disk

If you have the `md5sum` tool installed (standard on Linux; we suggest `conda install coreutils` to install this on macOS), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/
$ md5sum -c MD5SUM.txt
...
$ cd ..
```

There is no need to decompress the files.

Amplicon primers & reference sequences

A region of ITS1 was amplified using the ITS6/ITS7 primer pair (`GAAGGTGAAGTCGTAACAAGG` and `AGCGTTCTTCATCGATGTGC`) which bind the 5.8S rDNA, described here:

> Cooke *et al.* (2000) A molecular phylogeny of Phytophthora and related oomycetes. https://doi.org/10.1006/fgbi.2000.1202

The left primer (ITS6) matches the THAPBI PICT default, but their right primer (ITS7) matches about 60bp further downstream in *Phytophthora*. This means we can use THAPBI PICT default settings and get meaningful but blinkered results (for the subset of the data which our narrower primer set would have amplified, using a *Phytophthora* centric database).

In order to classify beyond *Phytophthora*, we need to build a THABPI PICT database including *Pythium* and *Phytopythium*. Redekar *et al.* (2019) Supplementary Table 3 provides a list of 1454 unique accessions and the species they assigned to it (not always the same as that listed on the NCBI record, as those annotations can change). Looking at those sequences, bar a handful they extend though the right primer. However, only about 50 have the left primer sequence included (depending how stringent you are), and the rest are also missing the next 32bp.

The ITS6 primer is situated within a highly conserved region, and the next 32bp is highly conserved, usually `TTTCCGTAGGTGAACCTGCGGAAGGATCATTA`. Unfortunately, the majority of published *Oomycetes* ITS1 sequences omit this. For the curated *Phytophthora* in the THAPBI PICT default database, we have inserted the expected sequence - and have yet to find a counter example. However, Redekar *et al.* (2019) took the other obvious choice, and remove it from their reads:

> *trimming extra bases from read1: an additional 32 bases from the 5 end of read1, which mapped to 18S segment, were trimmed as the oomycete ITS reference database does not include the 18S segment;*

We can do something similar in THAPBI PICT by treating this typically conserved 32bp region as part of the left primer - requiring it be present (while allowing some ambiguity) and removing it - leaving a shorter fragment which can be matched to a database built of those 1454 accessions.

Metadata

The provided file `metadata.tsv` has seven columns:

1. Source, "Reservoir", "River" or "Runoff"
2. Site, "A", "B", "C", ..., "M"
3. Process, "Filtration" or "Leaf baiting"
4. Period, "01" to "28"
5. Year-Month, "2015-04" to "2016-05" (given as "YYYY-MM" for sorting)
6. Sample, author's sample name, e.g. "OSU484"
7. Accession, assigned by the public archive, e.g. "SRR6303585"

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 7 -c 1,2,3,4,5,6
```

Argument `-t metadata.tsv` says to use this file for the metadata.

The `-x 7` argument indicates the filename stem can be found in column 7, Accession.

Argument `-c 1,2,3,4,5,6` says which columns to display and sort by (do not include the indexed column again). If for example the accession was listed first, it would be sorted on that, which is not helpful here. If you prefer to sort on site first, or by date before process, this should be straightforward.

We have not given a `-g` argument to assign colour bands in the Excel reports, so it will default to the first column in `-c`, meaning we get three coloured bands for "Reservoir", "River" and "Runoff".

Other files

Files `Redekar_et_al_2019_sup_table_3.tsv` (plain text tab separated table) and `Redekar_et_al_2019_sup_table_3.fasta` (FASTA format) are based on the Excel format Supplementary Table 3 from the paper.

4.2.2 Pipeline with defaults

Running thapbi-pict pipeline

First, we will run the THAPBI PICT pipeline command with largely default settings (including the default database and primers), other than including the metadata about the water samples. Note that this dataset has no blanks or negative controls, so we must trust the default minimum abundance threshold.

The key values which we will be changing later are the primers and database.

Assuming you have the FASTQ files in `raw_data/`, run the pipeline command as follows, and you should get the listed output report files:

```
$ mkdir -p intermediate_defaults/ summary/
$ thapbi_pict pipeline \
  -i raw_data/ -o summary/recycled-water-defaults \
  -s intermediate_defaults/ \
  -t metadata.tsv -x 7 -c 1,2,3,4,5,6
...
```

(continues on next page)

(continued from previous page)

```
onebp classifier assigned species/genus to 436 of 794 unique sequences from 1 files
Wrote summary/recycled-water-defaults.ITS1.samples.onebp.*
Wrote summary/recycled-water-defaults.ITS1.reads.onebp.*
...
$ ls -1 summary/recycled-water-defaults.*
summary/recycled-water-defaults.ITS1.onebp.tsv
summary/recycled-water-defaults.ITS1.reads.onebp.tsv
summary/recycled-water-defaults.ITS1.reads.onebp.xlsx
summary/recycled-water-defaults.ITS1.samples.onebp.tsv
summary/recycled-water-defaults.ITS1.samples.onebp.xlsx
summary/recycled-water-defaults.ITS1.tally.tsv
```

Here we used `-r` (or `--report`) to specify a different stem for the report filenames. The *sample metadata options* were described earlier – this is perhaps an idealised example in that `metadata.tsv` was created so that we add the first six columns the table (sorted in that order), where `-x 7` means index to the accession (filename prefix) in column seven.

Notice the output reported a taxonomic assignment for 431 of 794 unique sequences - that's 54%, but considerably higher if we consider the reads.

Results

We will compare and contrast the following four samples with the second run using different primers and a custom database. These were deliberately picked from the less diverse samples for clarity.

Here we pick out the four samples at the command line with `grep`, you can also look at the `recycled-water-defaults.ITS1.samples.onebp.xlsx` file in Excel:

```
$ cut -f 6,7,8 summary/recycled-water-defaults.ITS1.samples.onebp.tsv \
  | grep -E "(SRR6303586|SRR6303586|SRR6303588|SRR6303596|SRR6303948)"
OSU482       SRR6303588  Phytophthora chlamydospora, Phytophthora x stagnum(*), Unknown
OSU483       SRR6303586  Phytophthora chlamydospora, Phytophthora x stagnum(*)
OSU536.s203  SRR6303948  Phytophthora ramorum
OSU121       SRR6303596  Phytopythium (unknown species)
```

Three of these four have *Phytophthora* (and one with an unknown), while the fourth has *Phytopythium*. However, this is discarding all the reads which do not match the default *Phytophthora* centric primers.

4.2.3 Different primers

This example is based on the following paper from another research group:

- Redekar *et al.* (2019) Diversity of *Phytophthora*, *Pythium*, and *Phytopythium* species in recycled irrigation water in a container nursery. https://doi.org/10.1094/PBIOMES-10-18-0043-R

The worked example starts by running the pipeline command with *default settings*, which uses the default *Phytophthora* centric database and primers. We can do that because the tool's default database defines primers which target a subset of the longer amplicon amplified in this example dataset.

Now we will change the primer settings. Using the actual right primer will extend the *Phytophthora* FASTA sequences about 60bp (and accept many more non-*Phytophthora*). The left primer is actually the same, but to match the analysis and references from Redekar *et al.* (2019), we want to trim off the typically conserved 32bp fragment `TTTCCGTAGGTGAACCTGCGGAAGGATCATTA` from the start of each amplicon, which we can do by pretending this is part of the left primer.

The up-shot is by cropping about 32bp off the start, and adding about 60bp at the end, we will no longer get any matches against the default database with the default classifier (it is too strict, the matches are too distant).

This means before we can run the entire pipeline, we will need to build a custom database. We'll discuss the sequences which go into this database next, but this use `--marker ITS1-long` to name this alternative marker, and set `--left GAAGGTGAAGTCGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTA` and `--right AGCGTTCTTCATCGATGTGC` to declare the primers.

4.2.4 Building a custom database

We will build a database using the same public sequences as Redekar *et al.* (2019), using the accessions given in Supplementary Table 3 in Excel format.

We have taken their list of accessions and species names (ignoring voucher or isolate numbers), edited some punctuation to match the NCBI taxonomy, added some missing accession version suffixes, deduplicated, and made a simple tab-separated plain text table, with 1454 entries. In the setup instructions for this example you should have got a copy of this file, named `Redekar_et_al_2019_sup_table_3.tsv`, and a matching FASTA file `Redekar_et_al_2019_sup_table_3.fasta` which we will import into the new database.

This table is sorted alphabetically by species then accession, and starts:

```
$ head Redekar_et_al_2019_sup_table_3.tsv
<SEE TABLE EXCERPT BELOW>
```

You could also look at the TSV file in Excel:

HQ643082.1	Achlya ambisexualis
HQ643083.1	Achlya ambisexualis
HQ643084.1	Achlya americana
HQ643085.1	Achlya aquatica
HQ643086.1	Achlya bisexualis
HQ643087.1	Achlya bisexualis
HQ643088.1	Achlya bisexualis
HQ643089.1	Achlya caroliniana
HQ643090.1	Achlya colorata
HQ643091.1	Achlya colorata

Determining the species

Consider `FJ666127.1` which Redekar *et al.* (2019) listed as *Phytophthora aquimorbida* - yet at the time of writing, the file downloaded from https://www.ebi.ac.uk/ena/browser/api/fasta/FJ666127.1 is as follows, with a species name of *Phytophthora* sp. CCH-2009b:

```
>ENA|FJ666127|FJ666127.1 Phytophthora sp. CCH-2009b isolate 40A6 internal transcribed␣
↪spacer 1, partial sequence; 5.8S ribosomal RNA gene, complete sequence; and internal␣
↪transcribed spacer 2, partial sequence.
CCACACCTAAAAACTTTCCACGTGAACTGTCTGTGATGTTGGGGGGCTGCTGCTGCTGCT
TCGGTGGCGGCGTGCTCCCATCAAACGAGGCCCTGGGCTGCAAAGTCGGGGGTAGTAGTT
ACTTTTTGTAAACCCTTTTCCTGTATTTTCTGAATATACTGGGGGGACGAAAGTCTCTGC
TTTTAACTAGATAGCAACTTTCAGCAGTGGATGTCTAGGCTCGCACATCGATGAAGAACG
CTGCGAACTGCGATACGTAATGCGAATTGCAGGATTCAGTGAGTCATCGAAATTTTGAAC
GCATATTGCACTTCCGGGTTATGCCTGGGAGTATGCCTGTATCAGTGTCCGTACATCAAT
```

(continues on next page)

(continued from previous page)

```
CTTGGCTTCCTTCCTTCCGTGTAGTCGGTGGCGGGAACGCGCAGACGTGAAGTGTCTTGC
CTGTGGCTCCAGCTGTTGTTGGGGTGGTGTGGGCGAGTCCTTTGAAATGTAAGATACTGT
TCTTCTCTTTGCTGGAAAAGCGTGCGCTGTGTGGTTGTGGAGGCTGCCGTGGTGGCCAGT
CGGCGACTGACTTCGTGCTGATGCGTGTGGAGAGGCTCTGGATTCGCGGTATGGTTGGCT
TCGGCTGAACTTCTGCTTATTTGGGTGTCTTTTCGCTGCGTTGGCGTGTCGGGGTTGGTG
AACCGTAGTCATTTCGGCTTGGCTTTTGAACCGCGTGGCTGTAGCGCGAAGTATGGCGGC
TGCCTTTGTGGCGGCCGAGAGGACGACCTATTTGGGACGATTGTGCGGCCTCGTGCTGCA
TCTCAA
```

Notice that the species name runs into the general description, which is problematic. Unless THAPBI PICT has a pre-loaded taxonomy to use for validation, it has to use heuristics to split up this long string - which is not fully reliable.

If we look at https://www.ncbi.nlm.nih.gov/nucleotide/FJ666127.1 on the NCBI website, we see it in GenBank format which is a little different:

```
LOCUS       FJ666127                786 bp    DNA     linear   PLN 09-MAR-2009
DEFINITION  Phytophthora sp. CCH-2009b isolate 40A6 internal transcribed spacer
            1, partial sequence; 5.8S ribosomal RNA gene, complete sequence;
            and internal transcribed spacer 2, partial sequence.
ACCESSION   FJ666127
VERSION     FJ666127.1
KEYWORDS    .
SOURCE      Phytophthora aquimorbida
  ORGANISM  Phytophthora aquimorbida
            Eukaryota; Stramenopiles; Oomycetes; Peronosporales;
            Peronosporaceae; Phytophthora.
...
```

The NCBI metadata has the species *Phytophthora aquimorbida* separate from the author submitted description which starts with an older name, "Phytophthora sp. CCH-2009b" - which is in fact listed as an alias on the NCBI taxonomy database under taxonomy ID 611798.

THAPBI PICT offers two solutions. By default the *entire* FASTA description (after the identifier) is the species name, giving full control to the user.

However, `-c ncbi` switches on NCBI heuristics. This is best used with a pre-loaded NCBI taxonomy in the database for validation purposes. This tries as many words as possible from the NCBI style FASTA description in looking for a match in the NCBI taxonomy, including synonyms. If that fails and lax mode is used (`-x` or `--lax`), it falls back on heuristics to identify which part of the description is the species.

Species validation

THAPBI PICT by default validates imports against the NCBI taxonomy, and that includes support for known synonyms. This requires downloading the taxonomy files and running the `thapbi-pict load-tax` command.

The NCBI currently provide their taxonomy dump in two formats, old and new. THAPBI PICT supports both, we'll use the old format as the download is half the size - we only need the `names.dmp`, `nodes.dmp` and `merged.dmp` files:

```
$ curl -L -O https://ftp.ncbi.nih.gov/pub/taxonomy/taxdump_archive/taxdmp_2019-12-01.zip
...
$ unzip -n -d taxdmp_2019-12-01 taxdmp_2019-12-01.zip
...
$ ls -1 taxdmp_2019-12-01/*.dmp
taxdmp_2019-12-01/citations.dmp
```

(continues on next page)

(continued from previous page)

```
taxdmp_2019-12-01/delnodes.dmp
taxdmp_2019-12-01/division.dmp
taxdmp_2019-12-01/gencode.dmp
taxdmp_2019-12-01/merged.dmp
taxdmp_2019-12-01/names.dmp
taxdmp_2019-12-01/nodes.dmp
```

Building the database becomes a two-step process, first importing the taxonomy, and second importing the sequences.

If you are working with different organisms you will also need to set the `-a` or `--ancestors` option which defaults to NCBI taxonomy ID 4762 for *Oomycetes*.

Primer trimming

We have provided file `Redekar_et_al_2019_sup_table_3.fasta` which contains primer trimmed versions of the full sequences of each accession, plus the species name from `Redekar_et_al_2019_sup_table_3.tsv` which was based on those given in Redekar *et al.* (2019) Supplementary Table 3 but with some light curation to better match the NCBI usage. Note that matching sequences have been combined into single FASTA records with a semi-colon separated description.

The sequencing trimming ought to be very close to that used in the Redekar *et al.* (2019) paper's database. This file was constructed with a short Python script parsing the information in `Redekar_et_al_2019_sup_table_3.tsv` and the downloaded full sequences. Then `cutadapt -g GAAGGTGAAGTCGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTA ...` found and removed 64 left prefixes. This was followed by running `cutadapt -a GCACATCGATGAAGAACGCT ...` which trimmed 1439 sequences (99.9%) and warned that the "adapter" might be incomplete because the sequence preceding it was highly conserved. That left 1451 sequences, but with many duplicates. This was made non-redundant giving 841 unique sequences with de-duplicated entries recorded with semi-colon separated FASTA title lines.

Now, let's load the FASTA file into a new THAPBI PICT database with the NCBI taxonomy pre-loaded (which will enable synonym support), but not enforced (`-x` or `--lax` mode). We'll name the new marker "ITS1-long" and record the left and right primers which will be used later when processing the reads:

```
$ rm -rf Redekar_et_al_2019_sup_table_3.sqlite  # remove it if already there
$ thapbi_pict load-tax -d Redekar_et_al_2019_sup_table_3.sqlite -t taxdmp_2019-12-01/
...
$ thapbi_pict import -d Redekar_et_al_2019_sup_table_3.sqlite \
  --lax --sep ";" -i Redekar_et_al_2019_sup_table_3.fasta \
  --left GAAGGTGAAGTCGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTA \
  --right AGCGTTCTTCATCGATGTGC --marker ITS1-long
File Redekar_et_al_2019_sup_table_3.fasta had 841 sequences, of which 838 accepted.
Of 1451 potential entries, loaded 1451 entries, 0 failed parsing.
```

Just a few short sequences were rejected - giving in total 1451 entries. The vast majority are recorded with an NCBI taxid, just four exceptions (visible if you run the last command with `-v` or `--verbose`):

- *Phytophthora taxon aquatilis* from FJ666126.1, which the NCBI say should be *Phytophthora* sp. CCH-2009a
- *Phytophthora fragaefolia* from AB305065.1, which the NCBI say should be *Phytophthora fragariaefolia*.
- *Phytophthora citricola sensu stricto* from FJ560913.1, which the NCBI say should be just *Phytophthora citricola*.
- *Phytopythium sp. amazonianum* from HQ261725.1, which the NCBI say should be *Pythium* sp. 'amazonianum'.

None of these are clear cut (there were a lot more conflicts, mostly down to differences in punctuation, already addressed in preparing the TSV and FASTA file).

If you left off the `-x` (or `--lax`) option, those four would not have been imported into the database.

Taxonomic conflicts

The ITS1 region is not ideal as a barcode sequence. In the *Phytophthora* there are many cases where the same marker is shared by multiple species. The `thapbi_pict conflicts` command is provided to check for this, or worse – conflicts at genus level:

```
$ thapbi_pict conflicts -h
...
```

Let's run this on the custom database, with output to a file:

```
$ thapbi_pict conflicts -d Redekar_et_al_2019_sup_table_3.sqlite -o conflicts.tsv; echo
↪"(Return code $?)"
(Return code 3)
```

Command line tools use a non-zero return code by convention to indicate an error. Here we return the number of genus level conflicts, three, as can be seen by looking at the start of the plain text tab separated table output:

```
$ head -n 5 conflicts.tsv
#MD5                              Level    Conflicts
87e588784b04ba5f4538ff91acb50c0f  genus    Lagenidium;Pythium
9bb2ab5b9f88256516f2ae618c16a62e  genus    Brevilegnia;Globisporangium
ff35f216832110904cc6fd1c9def33fd  genus    Achlya;Saprolegnia
077ae505c0ad210aa4c071417a4f2f9a  species  Saprolegnia monilifera;Saprolegnia unispora
```

There are lots species level conflicts, some of which might be subspecies etc. However, more concerning is three genus level conflicts.

One way to see which accessions are a problem is filtering the dump command output (introduced properly in *Examining the database*), e.g.

```
$ thapbi_pict dump -d Redekar_et_al_2019_sup_table_3.sqlite \
  | cut -f 2-6 | grep 87e588784b04ba5f4538ff91acb50c0f
HQ643136.1  Lagenidium  caudatum   135481  87e588784b04ba5f4538ff91acb50c0f
HQ643539.1  Pythium     flevoense  289620  87e588784b04ba5f4538ff91acb50c0f
Wrote 1451 txt format entries
```

Some could be mislabelled, for `9bb2ab5b9f88256516f2ae618c16a62e` we see the vast majority are *Globisporangium ultimum* with just one sequence HQ643127.1 labelled as *Brevilegnia gracilis*:

```
$ thapbi_pict dump -d Redekar_et_al_2019_sup_table_3.sqlite \
  | cut -f 3-6 | grep 9bb2ab5b9f88256516f2ae618c16a62e \
  | sort | uniq -c | sed 's/^ *//g'
1 Brevilegnia       gracilis  944588   9bb2ab5b9f88256516f2ae618c16a62e
42 Globisporangium  ultimum   2052682  9bb2ab5b9f88256516f2ae618c16a62e
```

Checking the current NCBI annotation of these accessions does not suggest problems with recent taxonomy changes like *Phytopythium* vs *Pythium*.

Those assignments might have changed since this was written. Taxonomy is fluid, so if using any single authority, make sure to document which version (e.g. month and year for the NCBI taxonomy).

4.2.5 Specifying custom primers

Running prepare-reads step

We first ran the pipeline command with *default settings*, if you skipped that we can do just the reads now:

```
$ mkdir -p intermediate_defaults/
$ thapbi_pict prepare-reads -i raw_data/ -o intermediate_defaults/
...
$ ls -1 intermediate_defaults/ITS1/SRR*.fasta | wc -l
384
```

We then created a database from the Redekar *et al.* (2019) reference accessions with their primers. Now we can run the pipeline again with this, which will start by applying the prepare-reads step to the FASTQ files in `raw_data/`:

```
$ mkdir -p intermediate/
$ thapbi_pict prepare-reads -i raw_data/ -o intermediate_long/ \
  --db Redekar_et_al_2019_sup_table_3.sqlite
...
$ ls -1 intermediate_long/ITS1-long/SRR*.fasta | wc -l
384
```

Here the database says the left primer is `GAAGGTGAAGTCGTAACAAGG` (same as the THAPBI PICT default) plus `TTTCCGTAGGTGAACCTGCGGAAGGATCATTA` (conserved 32bp region), and that the right primer is `AGCGTTCTTCATCGATGTGC`. This has reverse complement `GCACATCGATGAAGAACGCT` and is found about 60bp downstream of the default right primer in *Phytophthora*, and should also match *Pythium* and *Phytopythium* species.

i.e. We should now find the *Phytophthora* FASTA sequences extracted are about 60 - 32 = 28bp longer, and many more non-*Phytophthora* are accepted.

Will now pick a couple of samples to compare and contrast with the first run. For clarity these examples are deliberately from the less diverse samples. The FASTA sequences have been line wrapped at 80bp for display.

Longer sequences

We will start with `SRR6303586` aka `OSU483`, a leaf-baiting sample from a reservoir. With the default primer trimming looking at the reads report, or the simpler sally table, focusing on just the one sample and filtering out non-zero counts:

```
$ tail -n +10 summary/recycled-water-defaults.ITS1.tally.tsv \
  | cut -f 3,386 | grep -v "^0"
<SEE TABLE BELOW>
```

You could instead select and filter on this column in Excel:

SRR	Sequence
35109	TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAAAACTTTCCACGT-GAACCGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCTGGCCTGTAATGGGTCGGCGTGCTGC
10271	TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAAAACTTTCCACGT-GAACCGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCTGGCCTGTAATGGGTCGGCGTGCTGC
580	TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAAACTTTCCACGTGAAC-CGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCTGGCCTGTAATGGGTCGGCGTGCTGCTGCT
157	TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAAACTTTCCACGTGAAC-CGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCTGGCCTGTAATGGGTCGGCGTGCTGCTGCT

Four very similar sequences (differing in the length of the poly-A run, seven is more common than six, and a C/T SNP towards the end), all matched to *Phytophthora chlamydospora* with THAPBI PICT's default settings.

With the new primer setting, which you can see listed at the start of the header, we again get four sequences passing the abundance threshold:

```
$ tail -n +10 summary/recycled-water-custom.ITS1-long.tally.tsv \
  | cut -f 3,386 | grep -v "^0"
<SEE TABLE BELOW>
```

As before, you may prefer to open this as a spreadsheet:

SRR	Sequence
3345	CCACACCTAAAAAAACTTTCCACGTGAACCGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCT
9729	CCACACCTAAAAAAACTTTCCACGTGAACCGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCT
545	CCACACCTAAAAAACTTTCCACGTGAACCGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCTG
143	CCACACCTAAAAAACTTTCCACGTGAACCGTATCAACCCCTTAAATTTGGGGGCTTGCTCGGCGGCGTGCGTGCTG

Again four very similar sequences, each as before but with the starting `TTTCCGTAGGTGAACCTGCGGAAGGATCATTA` removed, and instead extended by `GTGGGGACGAAAGTCTCTGCTTTTAACTAGATAGCAACTTTCAGCAGTGGATGTCTAGGCTC`.

The abundances are similar but slightly lower - there would have been some minor variation in trimmed regions which would have been pooled, so with less trimming we tend to get lower counts.

You can verify by NCBI BLAST online that the first and third (the `C` form) give perfect full length matches to published *Phytophthora chlamydospora*, while an exact match to the `T` forms has not been published at the time of writing (yet this occurs at good abundance in many of these samples).

Losing sequences

If you examine `SRR6303588` you will see a similar example, starting with five unique sequences (with one only just above the default abundance threshold), dropping to four unique sequences.

Finding *Pythium*

Now for a more interesting example, `SRR6303596` aka `OSU121`, another leaf baiting sample but from runoff water. With the defaults (using `grep` to omit the header):

```
$ tail -n +10 summary/recycled-water-defaults.ITS1.tally.tsv \
  | cut -f 13,386 | grep -v "^0"
<SEE TABLE BELOW>
```

As a table,

SRR	Sequence
953	TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAATCTTTCCACGT-GAATTGTTTTGCTGTACCTTTGGGCTTCGCCGTTGTCTTGTTCTTTTGTAAGAGAAAGGGGGAGGCGCGGTTGGAGC

There was a single sequence, with no matches (NCBI BLAST suggests this is *Phytopythium litorale*). Now with the revised primer settings this sequence is still present but only the second most abundant sequence:

```
$ tail -n +10 summary/recycled-water-custom.ITS1-long.tally.tsv \
  | cut -f 13,386 | grep -v "^0"
<SEE TABLE BELOW>
```

As a table, note this is sorted by global abundance:

SRR	Sequence
4050	CCACACCAAAAAAACTTTCCACGTGAACCGTTGTAACTATGTTCTGTGCTCTCTTCTCG-GAGAGAGCTGAACGAAGGTGGGCTGCTTAATTGTAGTCTGCCGATGTACTTTTAAACCCATTAAACTAATACTGAAC
878	CCACACCTAAAAATCTTTCCACGTGAATTGTTTTGCTGTACCTTTGGGCTTCGC-CGTTGTCTTGTTCTTTTGTAAGAGAAAGGGGGAGGCGCGGTTGGAGGCCATCAGGGGTGTGTTCGTCGCGGTTTGT
388	CCACACCAAAAAACTTTCCACGTGAACCGTTGTAACTATGTTCTGTGCTCTCTTCTCGGA-GAGAGCTGAACGAAGGTGGGCTGCTTAATTGTAGTCTGCCGATGTACTTTTAAACCCATTAAACTAATACTGAACTA
128	CCACACCAAAAAAACTTTCCACGTGAACCGTTGTAACTATGTTCTGTGCTCTCTTCTCG-GAGAGAGCTGAACGAAGGTGGGCTGCTTAATTGTAGTCTGCCGATGTACTTTTAAACCCATTAAACTAATACTGAAC
102	CCACACCAAAAAAACTTTCCACGTGAACCGTTGTAACTATGTTCTGTGCTCTCTTCTCG-GAGAGAGCTGAACGAAGGTGGGCTGCTTAATTGTAGTCTGCCGATGTACTTTTAAACCCATTAAACTAATACTGAAC
190	CCACACCAAAAAAACTTTCCACGTGAACCGTTGTAACTATGTTCTGTGCTCTCTTCTCG-GAGAGAGCTGAACGAAGGTGGGCTGCTTAATTGTAGTCTGCCGATGTACTTTTAAACCCATTAAACTAATACTGAAC

The probable *Phytopythium litorale* has been joined by five shorter and very similar sequences (differing by a handful of SNPs and a poly-A length change), which NCBI BLAST matches suggest are all *Pythium coloratum/dissotocum*.

Finding more

Another interesting example, `SRR6303948` aka `OSU536.s203`, from a runoff filtration sample. First with the default settings, a single unique sequence matching *Phytophthora ramorum*:

```
$ tail -n +10 summary/recycled-water-defaults.ITS1.tally.tsv \
  | cut -f 365,386 | grep -v "^0"
<SEE TABLE BELOW>
```

As a table,

SRR6	Sequence
1439	TTTCCGTAGGTGAACCTGCGGAAGGATCATTACCACACCTAAAAAACTTTCCACGTGAAC-CGTATCAAAACCCTTAGTTGGGGGCTTCTGTTCGGCTGGCTTCGGCTGGCTGGGCGGCGGCTCTATCATGGCGAG

Now with the revised primer settings, we get a further nine sequences - and the extended *Phytophthora ramorum* sequence drops to third most abundant:

```
$ tail -n +10 summary/recycled-water-custom.ITS1-long.tally.tsv \
  | cut -f 365,386 | grep -v "^0"
<SEE TABLE BELOW>
```

As a table, note this is sorted by global abundance:

SRR	Sequence
3287	CCACACCCGGGATCCTCGATCTTTCTCCTAGGTTAATTGTTGGGCCCTTTGAGGGTGGGC-CTTAGGTGCGCTCAAGGATTTTTTCCTGTCCCATGTAGCTTTACTTATTTTTTTGCCTGGGTAAATGATGGATTATTT
438	CCACACCAAAAAAACTTACCACGTGAATCTGTACTGTTTAGTTTTGTGCTGCGTTC-GAAAGGATGCGGCTAAACGAAGGTTGGCTTGATTACTTCGGTAATTAGGCTGGCTGATGTACTCTTTTAAACCCCTT
5329	CCACACCAAAAAAACACCCCACGTGAATTGTACTGTATGAGCTATGTGCTGCGGATTTCT-GCGGCTTAGCGAAGGTTTCGAAAGAGACCGATGTACTTTTAAACCCCTTTACATTACTGTCTGATAAATTACATTGC
144	CCACACCCGGGATCCTCGATCTTTCTCCTAGGTTAATTATTGGGCCCTTTGAGGGTGGGC-CTTAGGTGCGCTCAAGGATTTTTTCCTGTCCCATGTAGCTTTACTTATTTTTTTGCCTGGGTAAATGATGGATTATTT
230	AATCTATCACAATCCACACCTGTGAACTTGCTTGTTGGCCTCTGCATGTGCTTCGGTAT-GTGCAGGTTGAGCCGATCGGATTAACTTCTGGTCGGCTTGGGGCCTCAACCCAATCCTCGGATTGGTTTGGGGTCG
1319	CCACACCTAAAAAACTTTCCACGTGAACCGTATCAAAACCCTTAGTTGGGGGCTTCT-GTTCGGCTGGCTTCGGCTGGCTGGGCGGCGGCTCTATCATGGCGAGCGCTTGAGCCTTCGGGTCTGAGCTAGTAGC
224	CCACACCCGGGATCCTCGATCTTTCTCCTAGGTTAATTGTTTGGCCCTTTGAGGGTGGGC-CTTAGGTGCGCTCAAGGATTTTTTCCTGTCCCATGTAGCTTTACTTATTTTTTTGCCTGGGTAAATGATGGATTATTT
231	CCACACCCGGGATCCTCGATCTTTCTCCTAGGTTAATTGTTGGGCCCTTTGAGGGTGGGC-CTTAGGTGCGCTCAAGGATTTTTTCCTGTCCCATGTAGCTTTACTTATTTTTTTGCCTGGGTAAATGATGGATTATTT
102	CCACACCAAAAAAACACCCCACGTGAATTGTACTGTATGAGCTATGTGCTGCGGATTTCT-GCGGCTTAGCGAAGGTTTCGAAAGAGACCGATGTACTTTTAAACCCCTTTACATTACTGTCTGATAAATTACATTGC
189	CCACACCTAAAAACTTTCCACGTGAATCGTTCTATATAGCTTTGTGCTTTGCGGAAACGC-GAGGCTAAGCGAAGGATTAGCAAAGTAGTACTTCGGTGCGAAACACTTTTCCGATGTATTTTTCAAACCCTTTTACT

NCBI BLAST suggests some of the new sequences could be *Oomycetes*, but there are no very close matches - and some of the tenuous best matches include uncultured fungus, diatoms, green algae, and even green plants.

4.2.6 Examining the database

This example follows on from *Different primers*, and assumes you have used `thapbi_pict import` with the provided FASTA file (based on Supplementary Table 3 in Redekar *et al.* 2019), and created a THAPBI PICT database named `Redekar_et_al_2019_sup_table_3.sqlite`.

As the extension might suggest, this is an Sqlite v3 database, and can be examined directly at the command line if you are very curious. However, we will briefly review the provided commands within THAPBI PICT for checking a database.

Database export

The `thapbi_pict dump` command is intended for database export and/or answering simple queries without needing to use SQL to query the database. It defaults to giving plain text tab separated tables, but FASTA is also supported:

```
$ thapbi_pict dump -h
...
```

By default it outputs all the sequences, but you can do simple taxonomic filtering at genus or species level, for example:

```
$ thapbi_pict dump -d Redekar_et_al_2019_sup_table_3.sqlite \
  -g Phytophthora -s fallax -o P_fallax.tsv
Wrote 5 txt format entries to 'P_fallax.tsv'
$ cut -c 1-84 P_fallax.tsv
<SEE TABLE BELOW>
```

This gives a short table, with the sequence truncated for display:

#Marker	Identifier	Genus	Species	TaxID	MD5	Sequence
ITS1-long	DQ297398.1	Phytophthora	fallax	360399	693cf88b7f57bcc7a3532a6b7ff0268a	CCA
ITS1-long	HQ261557.1	Phytophthora	fallax	360399	693cf88b7f57bcc7a3532a6b7ff0268a	CCA
ITS1-long	HQ261558.1	Phytophthora	fallax	360399	693cf88b7f57bcc7a3532a6b7ff0268a	CCA
ITS1-long	HQ261559.1	Phytophthora	fallax	360399	693cf88b7f57bcc7a3532a6b7ff0268a	CCA
ITS1-long	DQ297392.1	Phytophthora	fallax	360399	da7ff4ae11bdb6cc2b8c2aea3937481f	CCA

The final columns give the amplicon marker sequence and its MD5 checksum.

Adding `-m` or `--minimal` to the command gives instead:

```
$ thapbi_pict dump -d Redekar_et_al_2019_sup_table_3.sqlite \
  -g Phytophthora -s fallax -o P_fallax.tsv -m
Wrote 2 txt format entries to 'P_fallax.tsv'
$ cut -c 1-56 P_fallax.tsv
<SEE TABLE BELOW>
```

Now the table only has one data row per unique marker sequence, again showing this with the sequence truncated:

#MD5	Species	Sequence
693cf88b7f57bcc7a3532a6b7ff0268a	Phytophthora fallax	CCA
da7ff4ae11bdb6cc2b8c2aea3937481f	Phytophthora fallax	CCA

Alternatively, we can ask for FASTA output:

```
$ thapbi_pict dump -d Redekar_et_al_2019_sup_table_3.sqlite \
  -g Phytophthora -s fallax -f fasta -o P_fallax.fasta
Wrote 2 fasta format entries to 'P_fallax.fasta'
```

This produces a short FASTA file as follows (with line wrapping added for display):

```
$ cat P_fallax.fasta
>DQ297398.1 Phytophthora fallax taxid=360399;HQ261557.1 Phytophthora fallax
taxid=360399;HQ261558.1 Phytophthora fallax taxid=360399;HQ261559.1 Phytophthora
fallax taxid=360399
CCACACCTAAAAAAATTCCACGTGAACTGTATTGTCAACCAAATTCGGGGATTCCTTGCTAGCGTGCCTTCGGGCGTGCC
GGTAGGTTGAGACCCATCAAACGAAAACATCGGCTGAAAGGTCGGAGCCAGTAGTTACCTTTGTAAACCCTTTACTAAAT
ACTGAAAAACTGTGGGGACGAAAGTCCTTGCTTTTACTAGATAGCAACTTTCAGCAGTGGATGTCTAGGCTC
>DQ297392.1 Phytophthora fallax taxid=360399
CCACACCTTAAAAAATTCCACGTGAACTGTATTGTCAACCAAATTCGGGGATTCCTTGCTAGCGTGCCTTCGGGCGTGCC
GGTAGGTTGAGACCCATCAAACGAAAACATCGGCTGAAAGGTCGGAGCCAGTAGTTACCTTTGTAAACCCTTTACTAAAT
ACTGAAAAACTGTGGGGACGAAAGTCCTTGCTTTTACTAGATAGCAACTTTCAGCAGTGGATGTCTAGGCTC
```

To be clear, each FASTA record is written as two potentially very long lines. The first title line consists of the FASTA new record > marker and then four semi-colon separated accessions with species. The sequence shared by those four

entries is given on the second line (without line breaks as markers tend not to be overly long, and it facilitates command line analysis/debugging).

Using the optional `-m` or `--minimal` switch changes the FASTA output to:

```
$ thapbi_pict dump -d Redekar_et_al_2019_sup_table_3.sqlite \
  -g Phytophthora -s fallax -f fasta -o P_fallax_minimal.fasta -m
Wrote 2 fasta format entries to 'P_fallax_minimal.fasta'
$ cat P_fallax_minimal.fasta
>693cf88b7f57bcc7a3532a6b7ff0268a Phytophthora fallax
CCACACCTAAAAAAATTCCACGTGAACTGTATTGTCAACCAAATTCGGGGATTCCTTGCTAGCGTGCCTTCGGGCGTGCC
GGTAGGTTGAGACCCATCAAACGAAAACATCGGCTGAAAGGTCGGAGCCAGTAGTTACCTTTGTAAACCCTTTACTAAAT
ACTGAAAAACTGTGGGGACGAAAGTCCTTGCTTTTACTAGATAGCAACTTTCAGCAGTGGATGTCTAGGCTC
>da7ff4ae11bdb6cc2b8c2aea3937481f Phytophthora fallax
CCACACCTTAAAAAATTCCACGTGAACTGTATTGTCAACCAAATTCGGGGATTCCTTGCTAGCGTGCCTTCGGGCGTGCC
GGTAGGTTGAGACCCATCAAACGAAAACATCGGCTGAAAGGTCGGAGCCAGTAGTTACCTTTGTAAACCCTTTACTAAAT
ACTGAAAAACTGTGGGGACGAAAGTCCTTGCTTTTACTAGATAGCAACTTTCAGCAGTGGATGTCTAGGCTC
```

This discards the original accessions and instead uses >, MD5 checksum, space, semi-colon separated list of taxonomic assignments, new line, sequence, new line. Again, there is deliberately no sequence line wrapping in the file itself.

Edit graph

In the worked example with the default database, we introduced the `edit-graph` command for use with CytoScape to examine the sequence space of the samples. It can also be run on a database alone provided you include the `-k` or `--marker` switch:

```
$ thapbi_pict edit-graph -k ITS1-long \
  -d Redekar_et_al_2019_sup_table_3.sqlite \
  -o Redekar_et_al_2019_sup_table_3.xgmml
Loaded 838 unique ITS1-long sequences from DB.
Computed Levenshtein edit distances.
Will draw 533 nodes with at least one edge (305 are isolated sequences).
```

Of the 838 unique sequences in the database, just over three hundred are isolated sequences (over 3bp edits away from anything else). The remaining five hundred plus give us an interesting edit distance graph.

Opening this in CytoScape the first thing that struck me was the largest two components are both for *Pythium regulare* - suggesting if these are truly all from one species that it has at least two distinct ITS1 markers in the genome?

Another use of this view would be to consider the genus conflicts reported by the `thapbi_pict conflicts` command - most of the handful of *Lagenidium* and *Brevilegnia* nodes are isolated.

4.2.7 Pipeline with custom database

Running thapbi-pict pipeline

Compared to the original worked example, we must specify our custom database (which contains the primer information, and matching primer trimmed entries):

```
$ mkdir -p intermediate_long/ summary/
$ thapbi_pict pipeline -i raw_data/ -s intermediate_long/ \
  -o summary/recycled-water-custom \
  -d Redekar_et_al_2019_sup_table_3.sqlite -m onebp \
```

(continues on next page)

(continued from previous page)

```
  -t metadata.tsv -x 7 -c 1,2,3,4,5,6
...
onebp classifier assigned species/genus to 529 of 3053 unique sequences from 1 files
Wrote summary/recycled-water-custom.ITS1-long.samples.onebp.*
Wrote summary/recycled-water-custom.ITS1-long.reads.onebp.*
...
$ ls -1 summary/recycled-water-custom.*.onebp.*
summary/recycled-water-custom.ITS1-long.onebp.tsv
summary/recycled-water-custom.ITS1-long.reads.onebp.tsv
summary/recycled-water-custom.ITS1-long.reads.onebp.xlsx
summary/recycled-water-custom.ITS1-long.samples.onebp.tsv
summary/recycled-water-custom.ITS1-long.samples.onebp.xlsx
```

Note the classifier method was set explicitly with `-m` (or `--method`), using the default of `onebp`. With the narrower set of *Phytophthora* sequences and comparatively well sampled database, that was a good default. Recall running with the *Phytophthora* defaults gave a taxonomic assignment for 2122757 of 2598566 reads - which was 82% of 2.6 million reads.

Here with our relatively sparse database, the `onebp` method is perhaps overly strict - only 17% of the unique sequences matched (529 of 3053 ASVs), although it is more like a third if we count the number of reads matched. However, with the different primer settings, we are examining over ten million reads (nearly four times as many), so we're doing about twice as well in terms of number of raw reads with a classification.

Naturally the more lenient or fuzzy `blast` based classifier makes even more matches:

```
$ thapbi_pict pipeline -i raw_data/ -s intermediate_long/ \
  -o summary/recycled-water-custom \
  -d Redekar_et_al_2019_sup_table_3.sqlite -m blast \
  -t metadata.tsv -x 7 -c 1,2,3,4,5,6
...
blast classifier assigned species/genus to 1036 of 3053 unique sequences from 1 files
Wrote summary/recycled-water-custom.ITS1-long.samples.blast.*
Wrote summary/recycled-water-custom.ITS1-long.reads.blast.*
...
$ ls -1 summary/recycled-water-custom.*.blast.*
summary/recycled-water-custom.ITS1-long.blast.tsv
summary/recycled-water-custom.ITS1-long.reads.blast.tsv
summary/recycled-water-custom.ITS1-long.reads.blast.xlsx
summary/recycled-water-custom.ITS1-long.samples.blast.tsv
summary/recycled-water-custom.ITS1-long.samples.blast.xlsx
```

Better, in that we are up to 34% of the unique sequences with a taxonomic assignment (1036 of 3053 ASVs). But how many of these are false positives? Sadly, we don't have any controls for this dataset in order to objectively assess the classifier performance of the various algorithm and database combinations.

However we can say that this database and indeed the published *Oomycetes* ITS1 sequences in general is relatively sparse outside *Phytophthora* (and even there, we as a community have room for improvement).

Results

We will focus on the same four low diversity samples for a brief comparison of the defaults, custom DB with `onebp`, and custom DB with `blast`.

Previously with the default DB and default `onebp` classifier:

```
$ cut -f 6,7,8 summary/recycled-water-defaults.ITS1.samples.onebp.tsv \
  | grep -E "(SRR6303586|SRR6303586|SRR6303588|SRR6303596|SRR6303948)"
OSU482       SRR6303588  Phytophthora chlamydospora, Phytophthora x stagnum(*), Unknown
OSU483       SRR6303586  Phytophthora chlamydospora, Phytophthora x stagnum(*)
OSU536.s203  SRR6303948  Phytophthora ramorum
OSU121       SRR6303596  Phytopythium (unknown species)
```

With the custom DB:

```
$ cut -f 6,7,8 summary/recycled-water-custom.ITS1-long.samples.onebp.tsv \
  | grep -E "(SRR6303586|SRR6303586|SRR6303588|SRR6303596|SRR6303948)"
OSU482       SRR6303588  Phytophthora chlamydospora, Phytophthora sp. CAL-2011b(*)
OSU483       SRR6303586  Phytophthora chlamydospora, Phytophthora sp. CAL-2011b(*)
OSU536.s203  SRR6303948  Phytophthora ramorum, Unknown
OSU121       SRR6303596  Phytopythium litorale, Pythium aff. diclinum(*), Pythium aff.␣
↪dictyosporum(*), Pythium aff. dissotocum(*), Pythium cf. dictyosporum(*), Pythium␣
↪coloratum(*), Pythium diclinum(*), Pythium dissotocum(*), Pythium lutarium, Pythium sp.
↪ CAL-2011f(*), Pythium sp. group F(*)
```

We get the same using the top BLAST hit:

```
$ cut -f 6,7,8 summary/recycled-water-custom.ITS1-long.samples.blast.tsv \
  | grep -E "(SRR6303586|SRR6303586|SRR6303588|SRR6303596|SRR6303948)"
OSU482       SRR6303588  Phytophthora chlamydospora, Phytophthora sp. CAL-2011b(*)
OSU483       SRR6303586  Phytophthora chlamydospora, Phytophthora sp. CAL-2011b(*)
OSU536.s203  SRR6303948  Phytophthora ramorum, Unknown
OSU121       SRR6303596  Phytopythium litorale, Pythium aff. diclinum(*), Pythium aff.␣
↪dictyosporum(*), Pythium aff. dissotocum(*), Pythium cf. dictyosporum(*), Pythium␣
↪coloratum(*), Pythium diclinum(*), Pythium dissotocum(*), Pythium lutarium, Pythium sp.
↪ CAL-2011f(*), Pythium sp. group F(*)
```

On this subset using `onebp` versus `blast` seems not to matter. The sample report does not go down to the sequences in each sample, for that you can use the reads report, or look at the intermediate FASTA files as discussed in the previous *primers* section.

The first two example differ due to the DB curation about exactly which *Phytophthora* is present. Sample `OSU121` aka `SRR6303596` went from one *Phytopythium litorale* sequence to being joined by a much more numerous *Pythium coloratum/dissotocum* sequence (plus some lower abundance variants of it). Likewise, `OSU536.s203` aka `SRR6303948` had one sequence for *Phytophthora ramorum*, but now has multiple unknown sequences.

4.3 Drained fish ponds 12S

This example uses a single 12S marker applied to fishing ponds which were later drained allowing identification of all the individual fish present:

> Muri *et al.* (2020) Read counts from environmental DNA (eDNA) metabarcoding reflect fish abundance and biomass in drained ponds. https://doi.org/10.3897/mbmg.4.56959

We provide a crude 12S database containing fish and off-target mammal and bird matches.

4.3.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/drained_ponds/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

FASTQ data

File `PRJNA638011.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata. Related file `metadata.tsv` combines this with metadata about the samples from the paper (see below).

Script `setup.sh` will download the raw FASTQ files for Muri *et al.* (2020) from https://www.ebi.ac.uk/ena/data/view/PRJNA638011 - you could also use https://www.ncbi.nlm.nih.gov/bioproject/PRJNA638011/

It will download 198 raw FASTQ files (99 pairs), taking about 550MB on disk

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/
$ md5sum -c MD5SUM.txt
$ cd ..
```

There is no need to decompress the files.

Amplicon primers & reference sequences

A region of 12S was amplified using a previously published primer pair (`ACTGGGATTAGATACCCC` and `TAGAACAGGCTCCTCTAG`) described here:

> Kelly *et al.* (2014) Understanding PCR processes to draw meaningful conclusions from environmental DNA studies. https://doi.org/10.1038/s41598-019-48546-x

This primer amplifies not just the fish of interest, but also birds and mammals (including human).

Rather than trying to use the same curated database from the University of Hull Evolutionary and Environmental Genomics Group, who also wrote metaBEAT (metaBarcoding and Environmental Analysis Tool) which was used in the paper, we provide a crudely curated database culled from stringent BLASTN matches in the NCBI NT database (search run with 100% query coverage and 99% identity, see provided `scripts/blast_to_fasta.py`), as file `NCBI_12S.fasta`. The `run.sh` script starts by loading this into a new THAPBI PICT database.

Metadata

The provided file `metadata.tsv` has ten columns, the first three are from `PRJNA638011.tsv` (ENA metadata) and the rest from the paper's Supplementary Table S2 - cross referenced on the sample name/alias:

1. run_accession, from ENA metadata, e.g. "SRR11949879"
2. sample_alias, from ENA metadata, e.g. "Lib3-M3-1F1"
3. sample_title, from ENA metadata, e.g. "MCE-Sample 1 filter 1"
4. samples, from Supplementary Table S2, e.g. "M3-1F1"
5. lake, from Table S2, e.g. "Middle_lake"
6. filter, from Table S2, e.g. "MCE"
7. treatment, from Table S2, e.g. "F1"
8. extracted, from Table S2, e.g. "Filter"
9. control, from Table S2, either "", "blank", "negative", or "positive"
10. date, from Table S2, e.g. "17.02.2017"

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 1 -c 5,6,7,8,9,10,4,3
```

Argument `-t metadata.tsv` says to use this file for the metadata.

The `-x 1` argument indicates the filename stem can be found in column 1, the ENA assigned run accession.

Argument `-c 5,6,7,8,9,10,4,3` says which columns to display and sort by (do not include the indexed column again). If for example the accession was listed first, it would be sorted on that, which is not helpful here. If you prefer to sort on filter first, that change should be straightforward.

We have not given a `-g` argument to assign colour bands in the Excel reports, so it will default to the first column in `-c`, meaning we get three coloured bands for "Middle_lake", "NA" (controls), and "New_lake".

Other files

Files `cichlid_control.known.tsv` and `negative_control.known.tsv` and are used in `setup.sh` to create `expected/*.known.tsv` entries for the positive and negative controls, including the blank controls.

12 fish species were translocated to New Lake, of which nine were also in the middle lake. Referring to the results text and Figure 1B, and pooling the two observed hybrids with a parent species, the expected species in the two lakes are as follows.

Middle lake and new lake both had:

- *Abramis brama*
- *Barbus barbus*
- *Carassius carassius*
- *Cyprinus carpio*
- *Perca fluviatilis*
- *Rutilus rutilus*
- *Scardinius erythrophthalmus*

- *Squalius cephalus*
- *Tinca tinca*

New lake only also had:

- *Acipenser* spp.
- *Ctenopharyngodon idella*
- *Silurus glanis*

File `middle_lake.known.tsv` lists the 9 species found in the middle lake, and `new_lake.known.tsv` lists the 12 species in the new lake (although not all fish are expected at all sites within each lake), and these are assigned to the remaining samples as `expected/*.known.tsv` by running `setup.sh`.

4.3.2 Presence and absence

Controls

Quoting the Muri *et al.* (2020) paper:

> A low-frequency noise threshold of 0.001 (0.1%) was applied across the dataset to reduce the probability of false positives arising from cross-contamination or tag-jumping (De Barba et al. 2014; Hänfling et al. 2016). Based on the level of contamination found in sampling/filtration blanks and PCR negatives, a second arbitrary threshold was applied and all records occurring with less than 50 reads assigned were removed.

To match the paper, this example uses `-a 50` for an absolute threshold of 50, and `-f 0.001` for a 0.1% sample specific factional threshold.

At this threshold, the 4 cichlid "positive" samples, 6 PCR "negative", and 8 "blank" controls are perfect - as far as the fish go. We do see unexpected human and chicken reads in the PCR negatives, and also ducks, cattle and pigs in the field "blanks":

```
$ grep -E "(^#|positive|negative|blank)" summary/drained_ponds.12S.samples.onebp.tsv |␣
↪cut -f 5,10-11,16,19
<SEE TABLE BELOW>
```

Or, filter/search `summary/drained_ponds.12S.samples.onebp.tsv` in Excel:

control	Sequencing sample	Classification summary	Threshold	Accepted
blank	SRR11949861	•	50	0
blank	SRR11949885	•	50	0
blank	SRR11949884	(Off-target) Homo sapiens, (Off-target) Sus scrofa	50	544
blank	SRR11949883	(Off-target) Bos taurus, (Off-target) Homo sapiens, (Off-target) Sus scrofa	50	1629
blank	SRR11949882	(Off-target) Anatidae (waterfowl)	50	61
blank	SRR11949881	(Off-target) Homo sapiens	50	56
blank	SRR11949880	(Off-target) Anatidae (waterfowl), (Off-target) Homo sapiens	50	436
blank	SRR11949834	(Off-target) Homo sapiens	50	175
negative	SRR11949908	•	50	0
negative	SRR11949907	(Off-target) Gallus gallus, (Off-target) Homo sapiens	50	606
negative	SRR11949851	•	50	0
negative	SRR11949850	•	50	0
negative	SRR11949838	(Off-target) Homo sapiens	50	71
negative	SRR11949837	(Off-target) Homo sapiens	50	356
positive	SRR11949836	Astatotilapia calliptera(*), Maylandia zebra(*)	50	39748
positive	SRR11949835	Astatotilapia calliptera(*), Maylandia zebra(*)	50	39244
positive	SRR11949906	Astatotilapia calliptera(*), Maylandia zebra(*)	65	62249
positive	SRR11949849	Astatotilapia calliptera(*), Maylandia zebra(*)	50	24566

Only in one sample (SRR11949906, a positive control) was the percentage based abundance threshold stricter than the

absolute threshold (65 not 50), and it still gives the highest number of reads.

Note that the positive samples only yield a single unique sequence (MD5 checksum `17dbc1c331d17cd075aabd6f710a039b`) which matches both the cichlid control species *Astatotilapia calliptera* and *Maylandia zebra*.

High level overview

Looking over `summary/drained_ponds.12S.samples.onebp.xlsx` in Excel, or the TSV equivalent of the sample report, there are some general trends visible.

First, as noted above the controls are extremely clean with just the expected cichlid species for the positive controls, and a few off-target matches in the PCR negatives and field blanks as noted above.

Next, both the Middle Lake Sterivex Ethanol Buffer, and Middle Lake Sterivex RNAlater Buffer are very clean. There are traces of human and waterfowl, and but only three of these buffer samples shows any fish (eg `SRR11949911` aka `5RNB`). In contrast, most of the Middle Lake Sterivex Longmire Buffer samples do give fish reads.

Controls and buffers aside, all the field samples gave Anatidae (waterfowl) matches, most had human. There were traces of other birds and mammals such as pig and dog - with most of the new lake samples showing sheep (*Ovis*).

As to the fish, we see strong signal in most samples for *Abramis brama*, *Carassius carassius*, *Cyprinus carpio*, *Rutilus rutilus* and *Tinca tinca*.

Expected Fish

This paper was selected as an example because something is known about the expected content of all the biological samples - the lakes were drained and all the fish identified, counted and weighed. However, we cannot expect all the species present to have left DNA at all the sampling points within their lake, but that is a useful approximation for assessing the classifier:

```
$ cut -f 1-5,9,11 summary/drained_ponds.12S.assess.onebp.tsv
<SEE TABLE BELOW>
```

You might prefer to open this in Excel:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	433	388	331	5877	0.55	0.624
(Off-target) Anatidae (waterfowl)	0	70	0	29	0.00	1.000
(Off-target) Apodemus	0	4	0	95	0.00	1.000
(Off-target) Ardea cinerea	0	11	0	88	0.00	1.000
(Off-target) Bos taurus	0	3	0	96	0.00	1.000
(Off-target) Canis lupus familiaris	0	7	0	92	0.00	1.000
(Off-target) Capra hircus	0	1	0	98	0.00	1.000
(Off-target) Columba	0	47	0	52	0.00	1.000
(Off-target) Gallinula chloropus	0	50	0	49	0.00	1.000
(Off-target) Gallus gallus	0	13	0	86	0.00	1.000
(Off-target) Homo sapiens	0	83	0	16	0.00	1.000
(Off-target) Ovis aries	0	17	0	82	0.00	1.000
(Off-target) Ovis dalli	0	1	0	98	0.00	1.000
(Off-target) Phalacrocorax carbo	0	25	0	74	0.00	1.000
(Off-target) Sturnus	0	3	0	96	0.00	1.000
(Off-target) Sus scrofa	0	16	0	83	0.00	1.000

continues on next page

Table 1 – continued from previous page

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
(Off-target) Turdus	0	7	0	92	0.00	1.000
Abramis brama	65	0	16	18	0.89	0.198
Acipenser spp.	0	0	9	90	0.00	1.000
Alburnus mossulensis	0	1	0	98	0.00	1.000
Astatotilapia calliptera	4	0	0	95	1.00	0.000
Barbus barbus	46	0	35	18	0.72	0.432
Carassius carassius	64	0	17	18	0.88	0.210
Ctenopharyngodon idella	3	15	6	75	0.22	0.875
Cyprinus carpio	61	0	20	18	0.86	0.247
Maylandia zebra	4	0	0	95	1.00	0.000
Perca fluviatilis	40	0	41	18	0.66	0.506
Pseudorasbora parva	0	2	0	97	0.00	1.000
Rutilus rutilus	63	0	18	18	0.88	0.222
Scardinius erythrophthalmus	6	0	75	18	0.14	0.926
Silurus glanis	9	0	0	90	1.00	0.000
Spinibarbus denticulatus	0	11	0	88	0.00	1.000
Squalidus gracilis	0	1	0	98	0.00	1.000
Squalius cephalus	6	0	75	18	0.14	0.926
Tinca tinca	62	0	19	18	0.87	0.235
OTHER 37 SPECIES IN DB	0	0	0	3663	0.00	0.000

False positives

We touched on the assorted "false positives" from the off-target 12S PCR amplification above. What is more interesting is the fish false positives. Let's look at these starting with the most false positives.

Ctenopharyngodon idella

First, many middle lake samples unexpectedly have *Ctenopharyngodon idella* (this is expected in the new lake samples). Why? They all stem from sequence `285edce3d193c92b1959e60bc130b518` which was matched to both *C. idella* and *Tinca tinca* (expected in both lakes):

```
>285edce3d193c92b1959e60bc130b518
ACTATGCTCAGCCATAAACCTAGACATCCACCTACAATTAAACGTCCGCCCGGGTACTACGAGCATTAGCTTGAAACCCA
AAGGACCTGACGGTGCCTTAGACCCCC
```

This is both a one base pair edit away from AY897013.1 etc as *C. idella*, and from AB218686.1 etc as *T. tinca*. Reviewing the NCBI BLAST matches both sets of species are supported from multiple complete mitochondrion genomes and a range of research groups. In the context of this experiment, we could infer for the four middle lake samples this sequence was *T. tinca*.

Spinibarbus denticulatus

Next, we see 16 samples with unexpected cyprinid fish *Spinibarbus denticulatus*. Referring to the read report, all are from a single sequence `4c53f6ed1ecdad3af2299999ec83d756` which has been matched perfectly to both this unexpected species and expected species *Carassius carassius*:

```
>4c53f6ed1ecdad3af2299999ec83d756
ACTATGCTCAGCCGTAAACTTAGACATCCTACTACAATAGATGTCCGCCAGGGTACTACGAGCATTAGCTTAAAACCCAA
AGGACCTGACGGTGTCTCAGACCCCC
```

Given the actual fish in these lakes have been taxonomically identified, we can safely dismiss this - and perhaps drop AP013335.1 *S. denticulatus* from the ad-hoc DB?

A similar choice was made in compiling the *ad hoc* database, dropping all the *Sander* sp. entries for the following sequence in favour of just *Perca fluviatilis* as the sole expected Percidae:

```
>7e88b1bdeff6b6a361cc2175f4f630fd
ACTATGCCTAGCCATAAACATTGGTAGCACACTACACCCACTACCCGCCTGGGAACTACGAGCATCAGCTTGAAACCCAA
AGGACTTGGCGGTGCTTTAGATCCAC
```

This was based on the authors' choice:

> All fish OTUs were identified to species level with the exceptions of records matching the family Percidae. Percidae records were manually assigned to *P. fluviatilis* as this was the only species of the family identified in the study area during fish relocation.

Pseudorasbora parva

We see two samples containing *Pseudorasbora parva*, the invasive species which prompted these fish ponds to be drained as a control measure. You can find this in the read report, at the command line:

```
$ grep -E "(Pseudorasbora parva|samples|predictions)" \
  summary/drained_ponds.12S.reads.onebp.tsv | cut -f 2,3,7,48,59
                                              samples          2LMB         3LMF
MD5                             onebp-predictions    Total-abundance  SRR11949854 ␣
↪SRR11949925
e819f3c222d6493572534fb6a5b7cda7  Pseudorasbora parva  520              323          197
```

Specifically we saw 323 reads in `SRR11949854` aka `2LMB` and 197 reads in `SRR11949925` aka `3LMF` - both middle lake Sterivex (STX) samples. Quoting the paper:

> *P. parva* reads found in two Middle Lake-STX samples (279 and 148 reads) were also excluded from further analyses as after eradication this species was not physically present at the site surveyed.

The exact counts differ, but referring to the paper's supplementary data the sample names match.

Other Fish

We also see one false positive for each of the two fish species *Alburnus mossulensis*, and *Squalidus gracilis*:

```
$ grep -E "(Alburnus mossulensis|samples|predictions)" \
  summary/drained_ponds.12S.reads.onebp.tsv | cut -f 2,3,7,25
                                                                    samples        ␣
↪M3-MF2
MD5                              onebp-predictions                  Total-abundance ␣
↪SRR11949859
916da937dccfd5d29502e83713e5d998  Abramis brama;Alburnus mossulensis  98              98
```

This sequence is ambiguous with equally good matches to expected species *Abramis brama*. Again, we might remove *Alburnus mossulensis* from the DB?

```
$ grep -E "(Squalidus gracilis|samples|predictions)" \
  summary/drained_ponds.12S.reads.onebp.tsv | cut -f 2,3,7,20
                                                 samples          M3-4F2
MD5                              onebp-predictions   Total-abundance  SRR11949871
c0d532d1c6f8ffff9c72ac4a1873151c  Squalidus gracilis  82               82
```

This sequence match is with AP011393.1 in the provided reference set.

False negatives

The classifier assessment shown above expected all the fish in each lake to be found at all the sites within that lake - an overly strong assertion which could explain many of the reported false negatives.

However, there is one clear false negative - neither this nor the original analysis found any *Acipenser* spp.

True positives

Rather than reviewing all of the true positives, I will note that in some cases we found more reads and thus declared a result in more samples. For example, we report *Barbus barbus* in 49 samples, versus:

> In addition, *Barbus barbus* was detected at two sites (202 reads), ...

We found *Scardinius erythrophthalmus* in six samples:

```
$ grep -E "(Scardinius erythrophthalmus|samples|predictions)" \
  summary/drained_ponds.12S.reads.onebp.tsv | cut -f 7,8,12,13,83,84,85
samples          M3-1F1       M3-5F1       M3-6F1       7RNF         8RNF         MRNF
Total-abundance  SRR11949879  SRR11949870  SRR11949868  SRR11949893  SRR11949886 ␣
↪SRR11949852
761              156          120          147          136          76           126
```

Quoting the original paper:

> The presence of *Scardinius erythrophthalmus* was found at two sites with a low number of reads (38 and 25 reads) and, therefore, removed after applying the filter threshold

In these cases at least, we are seeing much higher read counts. Given the supplementary data provided, it could be possible to plot the read counts from the two methods against each other.

Conclusion

While not in-depth, this hopefully demonstrates the THAPBI PICT could be meaningfully applied to this 12S dataset which was originally analysed with metaBEAT v0.97.11.

4.4 Fungal Mock Community ITS1 & 2

Here we consider mock communities of 19 fungal sequences (in both equal and staggered ratios), prepared with various protocols, and negative controls.

This example is based on the two amplicon sequencing libraries from this paper:

> Bakker (2018) A fungal mock community control for amplicon sequencing experiments. https://doi.org/10.1111/1755-0998.12760

The first library used a single primer set targeting ITS1, while the second library used two sets of primers targeting a different region of ITS1, and ITS2.

4.4.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/fungal_mock/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

FASTQ data

File `PRJNA377530.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata.

Script `setup.sh` will download the raw FASTQ files for Bakker (2018) from https://www.ebi.ac.uk/ena/data/view/PRJNA377530

It will download 122 raw FASTQ files (61 pairs), taking 346MB on disk.

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/AL1/
$ md5sum -c MD5SUM.txt
...
$ cd ../../
```


```
$ cd raw_data/AL1/
$ md5sum -c MD5SUM.txt
...
$ cd ../../
```


There is no need to decompress the files.

Amplicon primers & reference sequences

Amplicon library one (AL1) amplified a small region of ITS1 using primer pair BITS/B58S3 (`ACCTGCGGARGGATC` and `GAGATCCRTTGYTRAAAGTT`), as shown in the paper's supplementary Table S4.

Amplicon library two (AL2) amplified a larger region of ITS1 using primer pair ITS1f/ITS2 (`CTTGGTCATTTAGAGGAAGTAA` and `GCTGCGTTCTTCATCGATGC`), which includes the first library's target region entirely. Similar yields as per supplementary Table S4 vs S5.

Additionally, amplicon library two (AL2) amplified ITS2 using primer pair ITS3-KYO2 with ITS4-KYO3 (`GATGAAGAACGYAGYRAA` and `CTBTTVCCKCTTCACTCG`), with lower yields as per supplementary Table S5 vs S6.

The example must run THAPBI PICT twice. First using a single-marker database for AL1 using the BITS/B58S3 primers, and then with a dual-marker database for AL2 using the ITS1f/ITS2 and ITS3-KYO2/ITS4-KYO3 primers. In fact the example runs it third time, as we can also try the BITS/B58S3 primers on the second amplicon library, because they amplify a subregion of what the ITS1f/ITS2 pair amplify. See the primer discussion on the similar Redekar *et al.* (2019) worked example.

Files `ITS1.fasta` and `ITS2.fasta` were extracted from supplementary materials appendix S2, with the species name alone added to the FASTA titles (for input to `thapbi_pict import` with primer trimming).

Metadata

The amplicon specific files `metadata_AL1.tsv` and `metadata_AL2.tsv` are based on the metadata downloaded from the ENA, with some reformatting. The split into amplicon one and two was based on supplementary Tables S4, S5 and S6 (for the mock community samples) and reading the paper (for placing the negative controls).

They have seven columns:

1. Accession, assigned by the public archive, e.g. "SRR5314337"
2. MiSeq-name, author's filename stem, e.g. "FMockE.HC_S190"
3. Condition, based on original name without replicate suffix, e.g. "MockE_HC"
4. Replicate, numeric, e.g. "1"
5. Sample-type, either "fungal mock community" or "negative control"
6. Group, e.g. "even" or "staggered A"
7. Protocol, e.g. "high PCR cycle number" or "standard workflow"

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata_AL1.tsv -c 5,6,7,3,4,2 -x 1 -g 6
$ thapbi_pict ... -t metadata_AL2.tsv -c 5,6,7,3,4,2 -x 1 -g 6
```

Argument `-t metadata.tsv` says to use this file for the metadata.

Argument `-c 5,6,7,3,4,2` says which columns to display and sort by. This means Sample-type, Group, Protocol, Condition, Replicate, MiSeq Name. The purpose here is to group the samples logically (sorting on accession or MiSeq Name would not work), and suitable for group colouring.

Argument `-x 1` (default, so not needed) indicates the filename stem can be found in column 1, Accession. We might have downloaded the files and used the author original names, in which case `-x 2` ought to work.

Argument `-g 6` means assign colour bands using column 6, Group. This is used in the Excel reports.

Other files

The provided `negative_control.known.tsv` and `mock_community.known.tsv` files lists the expected species in the negative controls (none) and the mock community samples (the same 19 species, although not always in equal ratios).

Sub-folders under `intermediate/` are used for intermediate files, a folder for each amplicon library (AL1 and AL2) and primer-pair combination.

4.4.2 Community Edit Graphs

The sequence *Edit Graph* is very useful for understanding what came off the sequencer - although you may need to play with the thresholds to find a sweet spot for hiding the noise.

My main conclusion from the figures below is that the THAPBI PICT default `onebp` classifier is reasonable for these fungal communities markers. However, for the ITS1 marker *Fusarium* needs closer examination, and there should be even more database entries for *Rhizomucor irregularis*. You would of course also need to expand the database beyond the 19 species in the mock community to use these ITS1 or ITS2 fungal markers more generally.

Image generation

If you have loaded an XGMML network file from THAPBI PICT into Cytoscape, you can interactively select nodes based on the `Max-sample-abundance` attribute and hide or remove them. This is helpful for exploring what minimum threshold to use for drawing a clear edit graph, but this does not update the `Sample-count` and node sizes which are based on it.

For that you can re-run `thapbi_pict edit-graph` with the higher sample level minimum abundance setting (`-a` or `--abundance`). You do not need to regenerate the intermediate per-sample FASTA files unless you want to use a lower threshold.

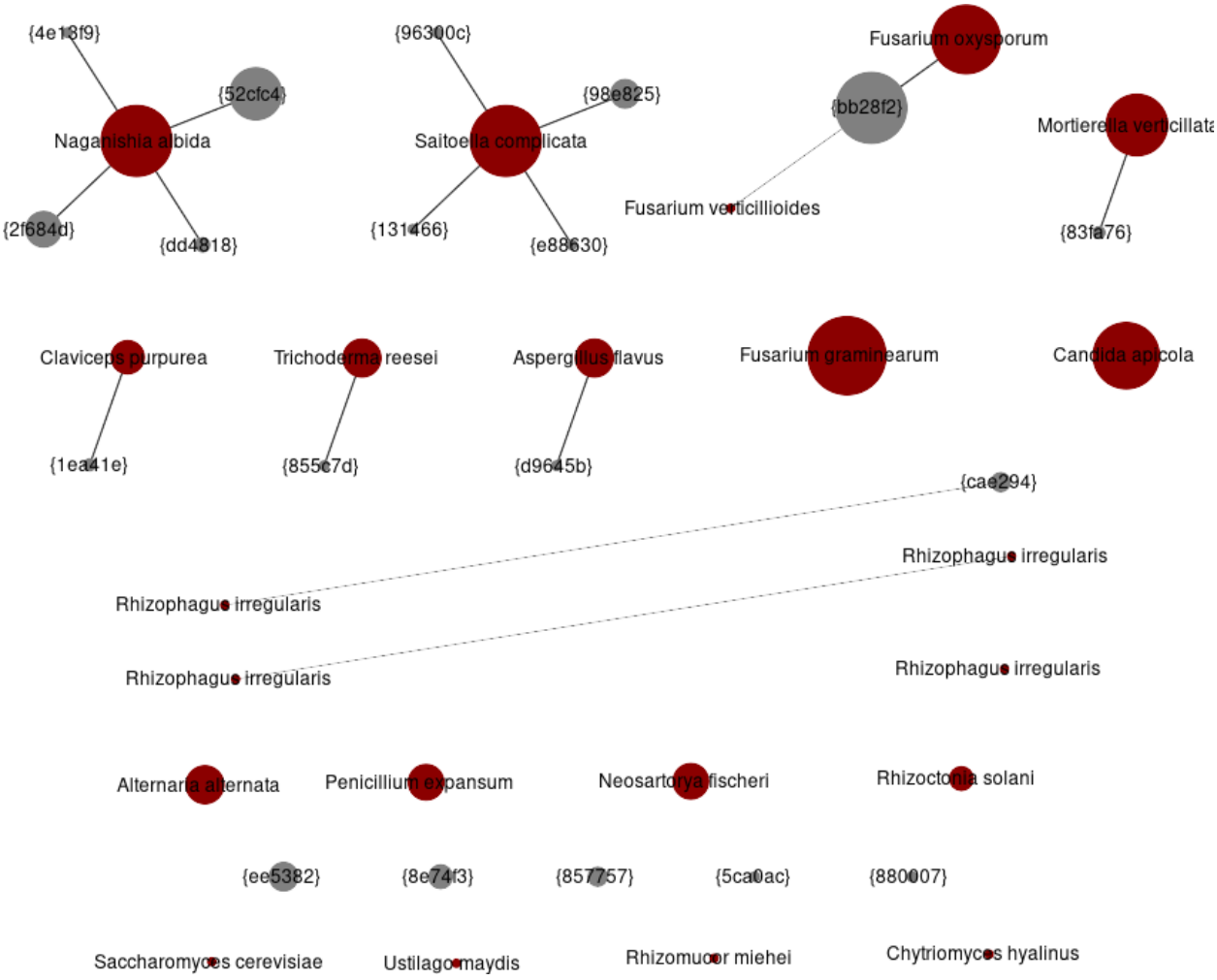
The following figures are from the example script `run.sh` which called `thapbi_pict edit-graph` with `-a 75`, meaning a unique sequence had to be in a sample from at least 75 reads to be considered. Using a lower value gives a much noiser picture (see the *Halo effect* discussed earlier).

Additionally this used `-k` (or `--marker`) to force including all of the database sequences (dark red nodes), as some did not appear in the samples (shown as the smallest dark red dots, typically the bottom row of the image). And, it used `-m -` (or `--method -`) to deliberately not label the nodes with the classifier output - only the data entries get a species label.

The XGMML files were loaded, automatically laid out using the "Perfuse Force Directed Layout" menu, manually adjusted to give a reasonably consistent node placement for comparison between the figures, and then images exported in SVG format (other formats are also supported including PDF and PNG).

Amplicon library one - ITS1

Starting with amplicon library one, where the BITS/B58S3 primers were used for a short fragment of ITS1.



This is from file `AL1.BITS_B58S3.edit-graph.a75.xgmml` created by `run.sh`.

The dark red nodes represent sequences in the database - given how this database was constructed to match the mock community, we would hope to see all the database entries represented in the samples. Some are missing at this abundance threshold (four bottom row entries *Saccharomyces cerevisiae*, *Ustilago maydis*, *Rhizomucor miehei* and *Chytriomyces hyalinus*, plus the four *Rhizomucor irregularis* entries shown across the middle.

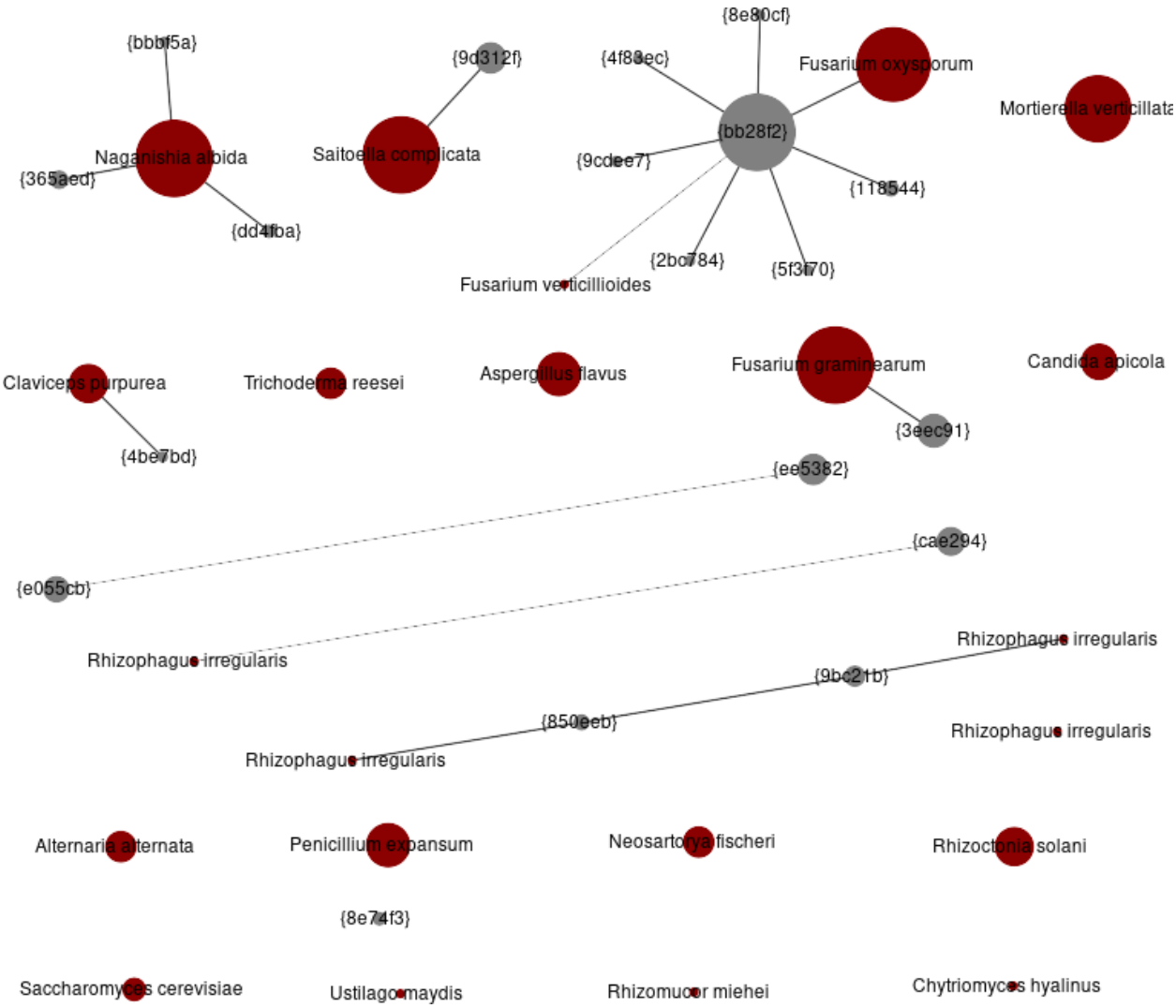
The large red nodes are the well represented community members, starting with *Naganishia albida* shown top left, which has four different 1bp variants some of which are large meaning they appear in many samples - you can see the sample counts in you load the XGMML file for this graph in Cytoscape. These are common enough to suggest they could be alternative versions of the ITS1 region in the genomes of these community members?

The (sometimes large) grey nodes not connected to a red node represent unwanted reads, likely contaminations discussed later.

In general each species is represented by a single connected component. The exceptions are *Rhizomucor irregularis* (multiple distantly related entries) and the *Fusarium*. The expected sequence for *Fusarium verticillioides* was not seen at all, however there are a great many copies one base away from the expected *Fusarium oxysporum* sequence (abbreviated MD5 checksum `bb28f2`, in full `bb28f2b57f8fddefe6e7b5d01eca8aea`). Is this perhaps coming from the *Fusarium verticillioides* strain?

Amplicon library two - ITS1

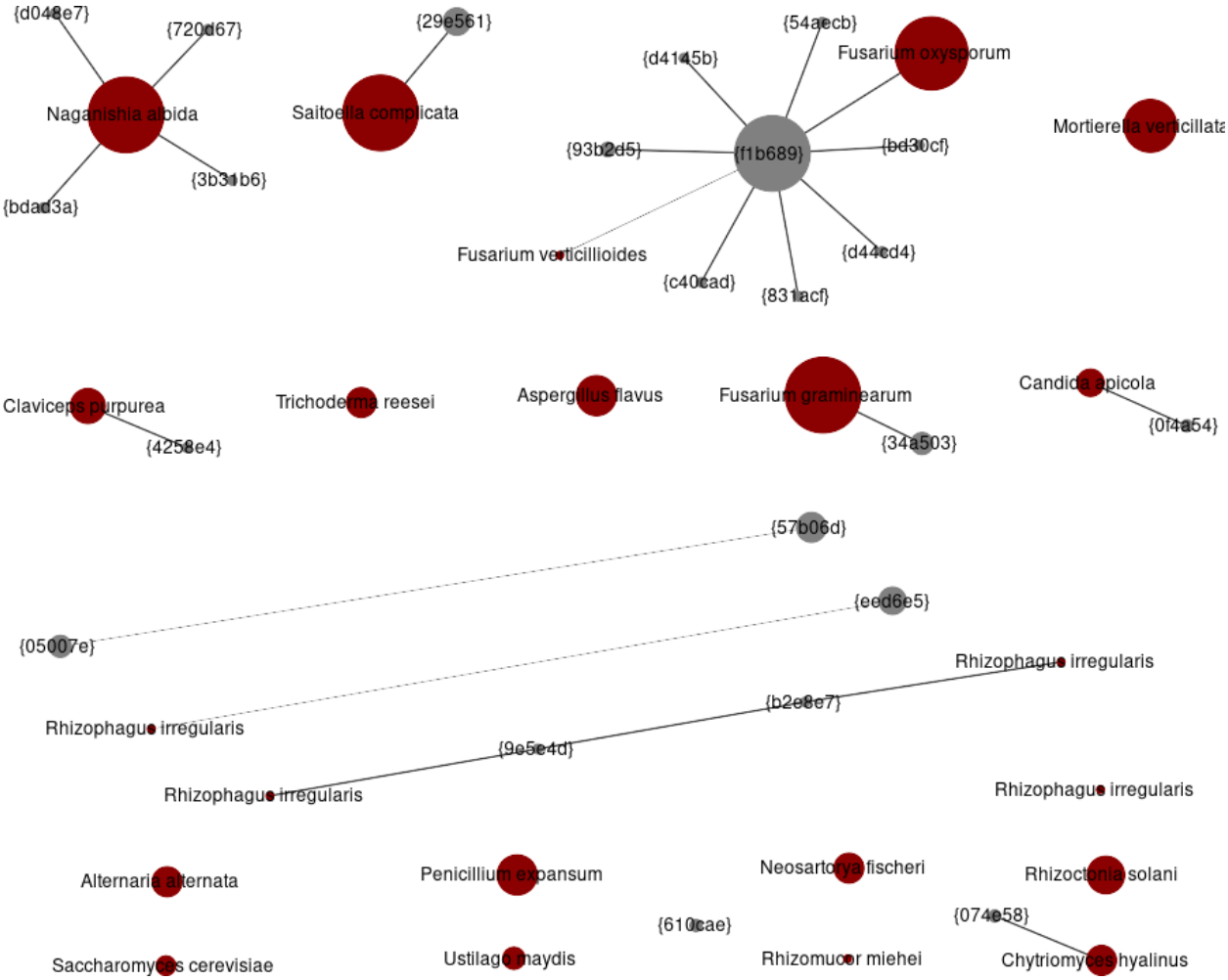
First, analysed using the same BITS/B58S3 primers as for ITS1 as in amplicon library one - the unique sequence MD5 checksums overlap with those seen in amplicon one:



This is from file `AL2.BITS_B58S3.edit-graph.a75.xgmml` created by `run.sh`.

Broadly the same as from amplicon library one, but notice the presence/absence patterns are different. Also there are more variants of the `bb28f2` *Fusarium*, and a pair of unexpected grey nodes 3bp apart (`e055cb` and `ee5482`, middle left, discussed below).

Now, using the actual primer pair, ITS1f/ITS2, which give a longer ITS1 fragment. Note that the sequences are extended so the checksums are different to those in the preceding images, but again broadly the same picture as the two images above:



This is from file `AL2.ITS1f_ITS2.edit-graph.a75.xgmml` created by `run.sh`.

The curious large grey node one edit away from *Fusarium oxysporum* has abbreviated MD5 checksum `f1b689`, or in full `f1b689ef7d0db7b0d303e9c9206ee5ad` (given in the XGMML node attributes). Referring back to the intermediate FASTA files or the read report, this does indeed represent the extended version of `bb28f2b57f8fddefe6e7b5d01eca8aea` with the first primer set:

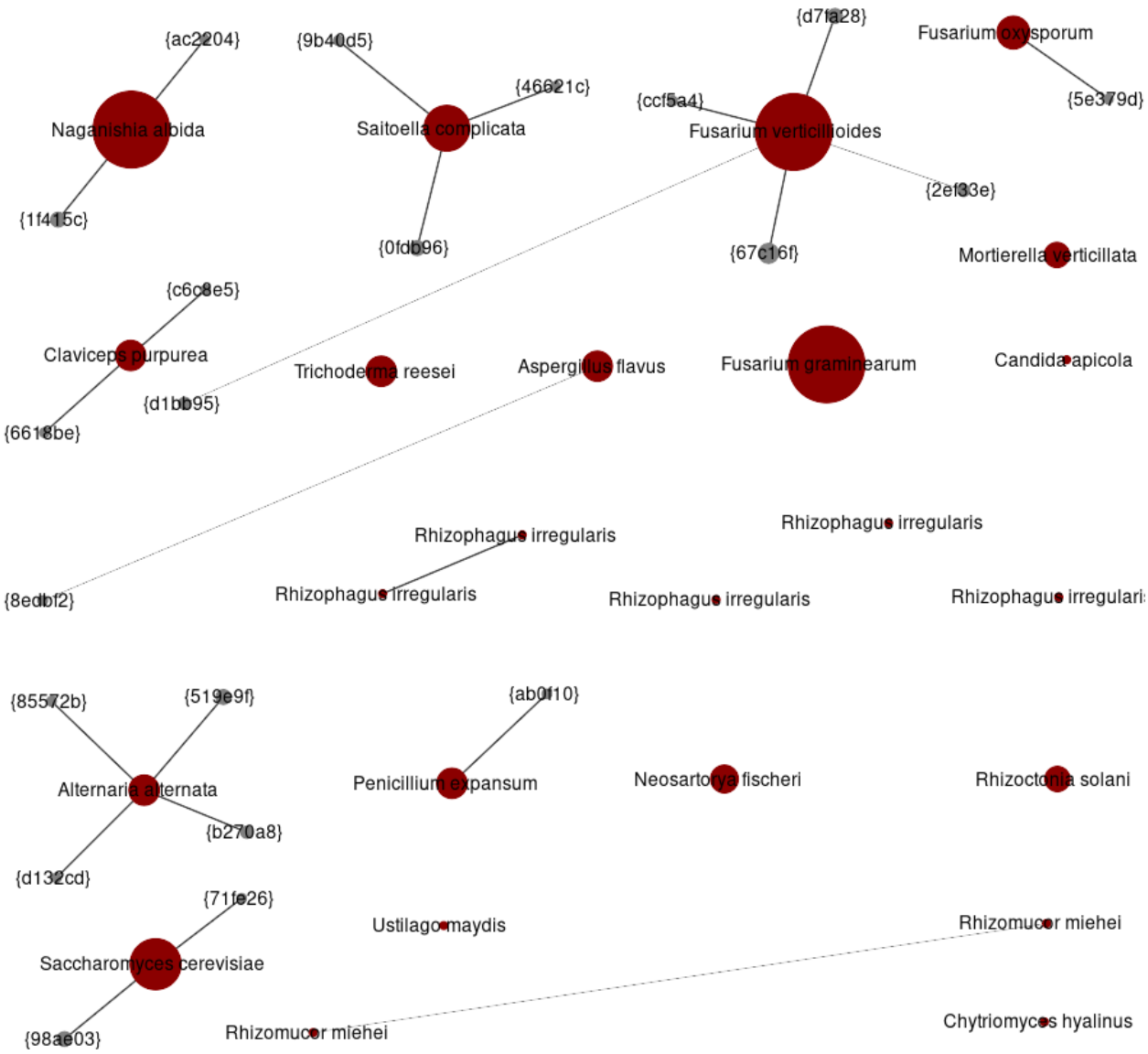
```
>bb28f2b57f8fddefe6e7b5d01eca8aea
ATTACCGAGTTTACAACTCCCAAACCCCTGTGAACATACCAATTGTTGCCTCGGCGGATCAGCCCGCTCCCGGTAAAACG
GGACGGCCCGCCAGAGGACCCCTAAACTCTGTTTCTATATGTAACTTCTGAGTAAAACCATAAATAAATCAA

>f1b689ef7d0db7b0d303e9c9206ee5ad
AAGTCGTAACAAGGTCTCCGTTGGTGAACCAGCGGAGGGATCATTACCGAGTTTACAACTCCCAAACCCCTGTGAACATA
CCAATTGTTGCCTCGGCGGATCAGCCCGCTCCCGGTAAAACGGGACGGCCCGCCAGAGGACCCCTAAACTCTGTTTCTAT
ATGTAACTTCTGAGTAAAACCATAAATAAATCAAAACTTTCAACAACGGATCTCTTGGTTCTG
```

Using an NCBI BLAST search, this exact sequence has been published from over a dozen different *Fusarium* species including *Fusarium oxysporum*, but not at the time of writing from *Fusarium verticillioides*.

The small pair of grey nodes 3bp apart (long diagonal line, middle left), `57b06d` and `05007e`, are the extended equivalents of `e055cb` and `ee5482` shown in the same place in the previous image. They seem to match *glomeromycetes*, perhaps from the *Rhizophagus* in the mock community.

Amplicon library two - ITS2

Finally, amplicon library two using the ITS3-KYO and ITS4-KYO3 primers for ITS2:



This is from file `AL2.ITS3-KYO2_ITS4-KYO3.edit-graph.a75.xgmml` created by `run.sh`.

Some more noteworthy changes to presence/absence, including much more *Saccharomyces cerevisiae* (still drawn bottom left). Also there are no unexpected grey nodes, and perhaps most interestingly from a species classification point of view, now the three *Fusarium* species fall into separate connected components.

4.4.3 Presence and absence

This example is a controlled setup where we know what the classifier ought ideally to report for every single sample.

We have replicated negative controls (which should have no marker sequences present), and plenty of positive controls (which should have the expected 19 species only).

Of course, just as in the original author's analysis, not everything we think was present is detected, and *vice versa*, lots of unwanted things are detected. These kinds of controls are perfect for discussing what to use as a minimum abundance threshold - how many reads do we need to declare a species as present in a sample?

Negative controls

If you have called the provided `setup.py` to download the FASTQ files and `run.py` to call THAPBI PICT, it would have used an optimistic minimum abundance threshold of 10 copies of each unique sequence (the default is a far more pesimitic 100).

This is not a good idea in general, but for your negative controls it can be interesting to deliberately set no threshold, and accept any sequence even if only supported by one read.

(Be sure to remove the intermediate FASTA files if you try this, as otherwise THAPBI PICT would not replace the older higher threshold files).

If you do this, just how bad are the contamination levels? These little tables were extracted manually from the sample level reports run with `-a 1` (accepting even sequences seen in only one read). The counts are the total number of reads in each sample, while max is the highest single sequence's abundance.

Amplicon library one, ITS1 using the BITS/B58S3 primer pair, samples replicated in duplicate:

Description	MiSeq-name	Accession	Count	Max
negative control from DNA extraction	NegDNAA_S163	SRR5314317	112	64
negative control from DNA extraction	NegDNAB_S175	SRR5314316	132	101
negative control from PCR step	NegPCRA_S187	SRR5314315	1153	1085
negative control from PCR step	NegPCRB_S104	SRR5314314	4343	3961

Amplicon library two, ITS1 using the ITS1f/ITS2 primer pair:

Description	MiSeq-name	Accession	Count	Max
negative control with GoTaq	NegCtlGoTq_S20	SRR5838526	2	1
negative control with Phusion	NegCtlPhGn_S30	SRR5838523	8	4
negative control with reAmp	NegCtlPrmp_S10	SRR5838524	9	1

Amplicon library two, ITS2 using the ITS3-KYO2 and ITS4-KYO3 primer pair:

Description	MiSeq-name	Accession	Count	Max
negative control with GoTaq	NegCtlGoTq_S20	SRR5838526	14	2
negative control with Phusion	NegCtlPhGn_S30	SRR5838523	17	4
negative control with PreAmp	NegCtlPrmp_S10	SRR5838524	5	1

Looking at these numbers the levels in amplicon library two are commendably low, at most four copies of any unique sequence - suggesting using a minimum threshold of 10 here is quite reasonable.

Hereafter we will assume the minimum abundance threshold of 10 was used, and you are encouraged to look at the sample or read level reports (e.g. in Excel) while following along with this discussion.

However, the levels in amplicon library one are cause for concern. Starting with the negative control from the DNA extraction (given a green background in the Excel reports), we see both replicates had two unwanted sequences. Look at `summary/AL1.BITS-B58S3.reads.onebp.xlsx` in Excel, or the TSV version at the command line:

```
$ cut -f 1,2,7,35,36 summary/AL1.BITS-B58S3.reads.onebp.tsv | grep -v
↪"[[:space:]]0[[:space:]]0$"
#                                                Sample-type      negative control    ␣
↪negative control
#                                                Group            from DNA extraction ␣
↪from DNA extraction
#                                                Protocol         standard workflow   ␣
↪standard workflow
#                                                Condition        Neg_DNA             ␣
↪Neg_DNA
#                                                Replicate        1                    2
#                                                MiSeq-name       NegDNAA_S163        ␣
↪NegDNAB_S175
#                                                Raw FASTQ        12564               ␣
↪16297
#                                                Flash            11641               ␣
↪15829
#                                                Cutadapt         112                  131
#                                                Threshold pool   AL1                  AL1
#                                                Threshold        10                   10
#                                                Control          Sample              ␣
↪Sample
#                                                Max non-spike    64                   100
#                                                Singletons       14                   17
#                                                Accepted         98                   110
#                                                Unique           2                    2
#Marker       MD5                                 Total-abundance  SRR5314317          ␣
↪SRR5314316
MAX or TOTAL  -                                  881219           64                   100
BITS-B58S3    d51507f661ebee38a85bec35b70b7ee1    47984            64                   100
BITS-B58S3    daadc4126b5747c43511bd3be0ea2438    34               34                   0
BITS-B58S3    e5b7a8b5dc0da33108cc8a881eb409f5    10               0                    10
```

Using a minimum of 10 has excluded lots of singletons etc here.

Both have `d51507f661ebee38a85bec35b70b7ee1` as their more common unwanted sequence, a perfect match to *Fusarium graminearum* in the mock community (classifier summary column omitted above for a clearer layout).

The lower abundance sequence `daadc4126b5747c43511bd3be0ea2438` gives perfect NCBI BLAST matches to several accessions of fungus *Wallemia muriae*, likewise `e5b7a8b5dc0da33108cc8a881eb409f5` gives perfect NCBI BLAST matches to *Wallemia muriae* and *Wallemia sebi*. They have no match from the classifier.

Moving on to the worst case, the negative control from the PCR reaction (given a pale blue background in the Excel reports). Again, look at the Excel file, or if working at the terminal:

```
$ cut -f 1,2,7,37,38 summary/AL1.BITS-B58S3.reads.onebp.tsv | grep -v
↪"[[:space:]]0[[:space:]]0$"
#                                                Sample-type      negative control  ␣
↪negative control
#                                                Group            from PCR step      from␣
↪PCR step
```

(continues on next page)

(continued from previous page)

```
#                                                Protocol         standard workflow␣
↪standard workflow
#                                                Condition        Neg_PCR            Neg_
↪PCR
#                                                Replicate        1                  2
#                                                MiSeq-name       NegPCRA_S187       ␣
↪NegPCRB_S104
#                                                Raw FASTQ        19406              7285
#                                                Flash            12140              6128
#                                                Cutadapt         1153               4340
#                                                Threshold pool   AL1                AL1
#                                                Threshold        10                 10
#                                                Control          Sample             ␣
↪Sample
#                                                Max non-spike    1085               3958
#                                                Singletons       42                 127
#                                                Accepted         1085               4014
#                                                Unique           1                  4
#Marker       MD5                                 Total-abundance  SRR5314315         ␣
↪SRR5314314
MAX or TOTAL  -                                  881219           1085               3958
BITS-B58S3    d51507f661ebee38a85bec35b70b7ee1    47984            1085               3958
BITS-B58S3    716f6111ac2ee192c23282e07d23078a    31294            0                  25
BITS-B58S3    5194a4ae3a27d987892a8fee7b1669b9    17               0                  17
BITS-B58S3    702929cef71042156acb3a28270d8831    14               0                  14
```

The minimum abundance excluded lots of singletons etc. The vast majority of those were slight variants of the dominant sequence, and can thus be explained as PCR noise.

Again, both samples have `d51507f661ebee38a85bec35b70b7ee1` as their main (or only) unwanted sequence above the threshold, a perfect match to *Fusarium graminearum* in the mock community. Additionally `716f6111ac2ee192c23282e07d23078a` matched *Mortierella verticillata* from the mock community.

Then `5194a4ae3a27d987892a8fee7b1669b9` gives perfect NCBI BLAST matches to fungus *Trichosporon asahii* and `702929cef71042156acb3a28270d8831` to fungus *Candida tropicalis*, which are unexpected contamination.

I concur with the author that the high levels of *Fusarium graminearum* are most likely cross-contamination from the mock-community samples:

> Negative control samples in this sequencing run displayed some contamination by *F. graminearum*. This taxon was represented at slightly, but not dramatically, higher than expected relative abundances in the mock community samples; some of the increase over expected relative abundance may have been related to cross-sample contamination.

Looking at the DNA extraction control alone, the THAPBI PICT default threshold of 100 seems reasonable. However, if we set that aside the likely *Fusarium graminearum* contamination, then the next worst contamination in any of these four controls is at 32 copies, so you might argue 100 is a little harsh?

Certainly I think for amplicon library one, a threshold of 10 is too low, but it could be defended for amplicon library two (where the controls had up to four copies of an unwanted sequence).

Missing positive controls

We will look at the ratios later, but were all 19 species in the mock community found? Perhaps the quickest way to answer this is to look at the classification assessment output. At the command line, looking at the BLAST based classifier as the most fuzzy of the three:

```
$ cut -f 1-5,9,11 summary/AL1.BITS-B58S3.assess.blast.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	345	5	168	71	0.80	0.334
Alternaria alternata	26	0	1	4	0.98	0.037
Aspergillus flavus	25	0	2	4	0.96	0.074
Candida apicola	27	0	0	4	1.00	0.000
Chytriomyces hyalinus	0	0	27	4	0.00	1.000
Claviceps purpurea	27	0	0	4	1.00	0.000
Fusarium graminearum	27	4	0	0	0.93	0.129
Fusarium oxysporum	27	0	0	4	1.00	0.000
Fusarium verticillioides	0	0	27	4	0.00	1.000
Mortierella verticillata	27	1	0	3	0.98	0.036
Naganishia albida	27	0	0	4	1.00	0.000
Neosartorya fischeri	24	0	3	4	0.94	0.111
Penicillium expansum	22	0	5	4	0.90	0.185
Rhizoctonia solani	19	0	8	4	0.83	0.296
Rhizomucor miehei	0	0	27	4	0.00	1.000
Rhizophagus irregularis	13	0	14	4	0.65	0.519
Saccharomyces cerevisiae	0	0	27	4	0.00	1.000
Saitoella complicata	27	0	0	4	1.00	0.000
Trichoderma reesei	27	0	0	4	1.00	0.000
Ustilago maydis	0	0	27	4	0.00	1.000

Or, open this plain text tab separated Excel.

Five expected species were never found (FN with zero true positives) at the 10 reads abundance threshold: *Chytriomyces hyalinus*, *Fusarium verticillioides*, *Rhizomucor miehei*, *Saccharomyces cerevisiae* and *Ustilago maydis*.

The author wrote:

> Two of the expected 19 phylotypes, *Fusarium verticillioides* and *Saccharomyces cerevisiae*, were not detected in any of the samples. A large number of reads, presumably including many *F. verticillioides* reads, were binned into a phylotype as unclassified *Fusarium*. The primers used in ITS1 amplification for this sequencing library match the rRNA gene sequence of *S. cerevisiae*. However, the expected ITS1 amplicon length is 402 bases for this taxon, compared to a range of 141-330 bases across the remaining taxa in the mock community. Examining the data at earlier stages of processing revealed that *S. cerevisiae* was originally represented in the data set, but was completely removed during quality screening (Table S3).
>
> *Chytriomyes hyalinus*, *Rhizomucor miehei* and *Ustilago maydis* were detected at dramatically lower abundances than expected. Each of these taxa possesses sequence mismatches compared to the PCR primers that were used. The number of mismatches to the forward and reverse primers was as follows: for *C. hyalinus*, 2 and 1; for *R. miehei*, 0 and 2; and for *U. maydis*, 2 and 1. Thus, selection against these taxa may have been due to primer annealing efficiency.

That's pretty consistent (we've talked about *Fusarium verticillioides* earlier), and suggests using a minimum abundance threshold of 10 in THAPBI PICT is a little stricter that the author's pipeline.

Moving on to the second amplicon library, the larger ITS1 marker using the ITS1f/ITS2 primer is more successful:

```
$ cut -f 1-5,9,11 summary/AL2.ITS1f-ITS2.assess.blast.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	398	0	115	57	0.87	0.224
Alternaria alternata	23	0	4	3	0.92	0.148
Aspergillus flavus	27	0	0	3	1.00	0.000
Candida apicola	12	0	15	3	0.62	0.556
Chytriomyces hyalinus	25	0	2	3	0.96	0.074
Claviceps purpurea	27	0	0	3	1.00	0.000
Fusarium graminearum	27	0	0	3	1.00	0.000
Fusarium oxysporum	27	0	0	3	1.00	0.000
Fusarium verticillioides	12	0	15	3	0.62	0.556
Mortierella verticillata	27	0	0	3	1.00	0.000
Naganishia albida	27	0	0	3	1.00	0.000
Neosartorya fischeri	23	0	4	3	0.92	0.148
Penicillium expansum	24	0	3	3	0.94	0.111
Rhizoctonia solani	24	0	3	3	0.94	0.111
Rhizomucor miehei	4	0	23	3	0.26	0.852
Rhizophagus irregularis	11	0	16	3	0.58	0.593
Saccharomyces cerevisiae	9	0	18	3	0.50	0.667
Saitoella complicata	27	0	0	3	1.00	0.000
Trichoderma reesei	25	0	2	3	0.96	0.074
Ustilago maydis	17	0	10	3	0.77	0.370

Everything was found, although *Rhizomucor miehei* in particular found rarely, followed by *Saccharomyces cerevisiae*. The original author wrote:

> The ITS1 data set yielded 18 of the expected 19 taxa (Tables S3, S5); as in the first library, no reads were classified as *F. verticillioides*, although many reads were placed in unclassified Fusarium. *Rhizomucor miehei* and *S. cerevisiae* were substantially underrepresented. Compared to primers ITS1f and ITS2, *R. miehei* had three mismatches in the forward and two mismatches in the reverse. *Saccharomyces cerevisiae* had one mismatch in the forward primer and again likely suffered negative bias associated with amplicon length (Table 3) and low sequence quality (Table S3).

Again, broad agreement here, with the problem of *Fusarium verticillioides* discussed earlier.

And finally, amplicon library two for ITS2 using the ITS3-KYO2 and ITS4-KYO3 primers:

```
$ cut -f 1-5,9,11 summary/AL2.ITS3-KYO2-ITS4-KYO3.assess.blast.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	313	0	200	57	0.76	0.390
Alternaria alternata	16	0	11	3	0.74	0.407
Aspergillus flavus	24	0	3	3	0.94	0.111
Candida apicola	0	0	27	3	0.00	1.000
Chytriomyces hyalinus	0	0	27	3	0.00	1.000
Claviceps purpurea	23	0	4	3	0.92	0.148
Fusarium graminearum	27	0	0	3	1.00	0.000
Fusarium oxysporum	27	0	0	3	1.00	0.000
Fusarium verticillioides	27	0	0	3	1.00	0.000
Mortierella verticillata	12	0	15	3	0.62	0.556
Naganishia albida	27	0	0	3	1.00	0.000
Neosartorya fischeri	16	0	11	3	0.74	0.407
Penicillium expansum	23	0	4	3	0.92	0.148
Rhizoctonia solani	11	0	16	3	0.58	0.593
Rhizomucor miehei	0	0	27	3	0.00	1.000
Rhizophagus irregularis	5	0	22	3	0.31	0.815
Saccharomyces cerevisiae	27	0	0	3	1.00	0.000
Saitoella complicata	26	0	1	3	0.98	0.037
Trichoderma reesei	22	0	5	3	0.90	0.185
Ustilago maydis	0	0	27	3	0.00	1.000

This time we're missing *Candida apicola*, *Chytriomyces hyalinus*, *Rhizomucor miehei* and *Ustilago maydis*.

This too is in board agreement with the original author, although *Candida apicola* must have just dipped below our abundance threshold.

> Different amplification biases were evident between the ITS1 and ITS2 loci. In the ITS2 data set, only 16 of the 19 taxa that were present could be detected; *C. hyalinus*, *R. miehei* and *U. maydis* were not observed (Tables S3, S6). … *Rhizomucor miehei* has one mismatch to the forward primer and three mismatches to the reverse primer. While neither *C. hyalinus* nor *U. maydis* have sequence mismatches compared to the primers, these two taxa have longer ITS2 amplicons than any others in the mock community (Table 3). These two taxa were originally represented with a small number of reads in the raw data, but were completely removed during quality screening (Table S3). *Candida apicola*, which possesses two mismatches to the reverse primer for this amplicon, was detected at substantially lower than expected frequencies (Figure 7; Figures S5, S6).

So, using THAPBI PICT on these amplicon datasets with a minimum abundance threshold of 10 gives broad agreement with the original analysis.

4.4.4 Unexpected sequences

In the previous section, we highlighted several unexpected contaminants in the negative controls which could not be explained as cross-contamination from the mock community. Likewise the read reports show plenty of unassigned sequences, things which did not match the very narrow databases built from `ITS1.fasta` or `ITS2.fasta` containing markers expected from the mock community *only*.

Some unexpected sequences might reflect additional alternative copies of ITS1 or ITS2 in the genomes. Others are likely external contamination - after all there are fungi practically everywhere. This seems to have happened on amplicon library one in the high PCR cycle negative control at least. Meanwhile, amplicon library two does not have any obvious external contamination.

Amplicon library one - ITS1 (BITS/B58S3)

From the first amplicon library for ITS1 we saw the following sequences in the negative controls (and by chance, not in any mock community samples) - shown here with their highest single sample abundance, which supports using a minimum abundance threshold higher than 10:

MD5 checksum	Max	Species
daadc4126b5747c43511bd3be0ea2438	32	*Wallemia muriae*
e5b7a8b5dc0da33108cc8a881eb409f5	10	*Wallemia muriae*; *Wallemia sebi*
5194a4ae3a27d987892a8fee7b1669b9	17	*Trichosporon asahii*
702929cef71042156acb3a28270d8831	14	*Candida tropicalis*

Here are the reads from entries with a maximum sample abundance over 75 which the `onebp` and in some cases `blast` based classifier failed to match, along with the most likely match from reviewing an online NCBI BLAST search. You can easily extract these entries (and their sequences) from the bottom of the `summary/AL1_BITS_B58S3.reads.*.tsv` files:

MD5 checksum	Max	Species
5ca0acd7dd9d76fdd32c61c13ca5c881	4562	*Epicoccum nigrum*; *Epicoccum layuense*
ee5382b80607f0f052a3ad3c4e87d0ce	575	*glomeromycetes*, perhaps *Rhizophagus*
880007c5a18be69c3f444efd144fc450	236	*Ascochyta* or *Neoascochyta*?
8e74f38b058222c58943fc6211d277fe	149	*Fusarium*
cae29429b90fc6539c440a140494aa25	114	*glomeromycetes*, perhaps *Rhizophagus*
85775735614d45d056ce5f1b67f8d2b2	109	*Fusarium*

The sequence with the top abundance, `5ca0acd7dd9d76fdd32c61c13ca5c881`, perfectly matches fungus *Epicoccum nigrum* and *Epicoccum layuense*. Present at low levels in multiple samples, this was the dominant sequence in `SRR5314339` aka `FMockE.HC1_S178`, which was a *high PCR cycle number* replicate of the even mixture. Perhaps this was a stray fragment of *Epicoccum* which by chance was amplified early in the PCR? This example was not highlighted in the original paper, but is exactly the kind of thing you should worry about with a high PCR cycle number.

Next `ee5382b80607f0f052a3ad3c4e87d0ce` and the less abundant sequence `cae29429b90fc6539c440a140494aa25` looks like *glomeromycetes*, perhaps *Rhizophagus* (from the mock community), but could be from a *Glomus* species. Using the `blast` classifier and the minimal curated reference set matches this to *Rhizophagus irregularis*, but the situation would be ambiguous in a more complete database.

Sequence `880007c5a18be69c3f444efd144fc450` has perfect matches to lots of unclassified fungi, and conflicting perfect matches including *Ascochyta* or *Neoascochyta*. This was seen only in the high PCR cycle number sample `SRR5314339` as above.

Next `8e74f38b058222c58943fc6211d277fe` and `85775735614d45d056ce5f1b67f8d2b2` have good BLAST matches to several different *Fusarium* species, so could also be from the mock community.

You can find all six of these sequence on the edit-graph, most as isolated grey nodes along the bottom except `cae29429b90fc6539c440a140494aa25` which is 3bp away from *Rhizophagus irregularis* and linked to it with a dashed line.

So some of the ITS1 sequences in amplicon library one are likely external contamination - particularly with the high PCR cycle negative control (which was likely included exactly because of this risk).

Amplicon library two - ITS1 (ITS1f/ITS2)

Using our `blast` classifier with the 19 species database, everything was assigned a match. The default `onebp` classifier was stricter. For example while the very common `f1b689ef7d0db7b0d303e9c9206ee5ad` (which with the BITS/B58S3 primers gave `bb28f2b57f8fddefe6e7b5d01eca8aea`) was matched to *Fusarium oxysporum*, all the variations of this were too far away from the database entries for a match.

These primers amplified a larger fragment to that in amplicon library one. Focusing on those with a sample-abundance over 75 (as in the edit-graphs) which the `onebp` classifier did not match to the curated reference set:

Long sequence MD5 (ITS1f/ITS2).	Max	Species
`57b06dff740b38bd6a0375abd9db3972`	640	*glomeromycetes*, perhaps *Rhizophagus*
`eed6e5c3881a233cca219f7ffd886bbe`	315	*glomeromycetes*, perhaps *Rhizophagus*
`05007e829ab71427b49743994a14105f`	154	*glomeromycetes*, perhaps *Rhizophagus*
`93b2d56429637947243e1b5d54a065cf`	132	*Fusarium*
`610caedb1a5699836310fce9dbb9c5fa`	96	*Fusarium*
`54aecb27334809f56b7f940b9ca060a3`	93	*Fusarium*
`bd30cf52b7031ddd96e3d7588c1f0e1c`	90	*Fusarium*
`c40cad2530d633430c3805be3740c9a4`	88	*Fusarium*
`d44cd471b11f15e2e42070806737e5d1`	86	*Fusarium*
`831acf596cca4ef840c5543d82e23d16`	82	*Fusarium*
`d4145ba9e3ed6c8c2138ed15b147152d`	81	*Fusarium*

You can find all of these sequence on the edit-graph, most of those labelled as likely *Fusarium* are a 1bp edit away from large grey node `f1b689` top left (except `610caedb1a5699836310fce9dbb9c5fa` which is an isolated node placed bottom middle). Those labelled *glomeromycetes* are in the middle near, and in once case connected to, a dark red *Rhizophagus irregularis* node.

i.e. None of the ITS1 sequences in amplicon library two are clear cut external contamination.

Amplicon library two - ITS2

Finally, amplicon library two using the ITS3-KYO and ITS4-KYO3 primers for ITS2. Again, the `blast` based classifier matched everything to an entry in the mock community database. The stricter `onebp` classifier assigned most reads. Here are those few it failed to match with a maximum read abundance over 75:

MD5 checksum	Max	Species
`d1bb95fff4a7e9958fa3c7f13cc51343`	211	*Fusarium*
`2ef33e6acd8079d729b81d24b91fcf88`	133	*Fusarium*
`8edbf2c168b11f910458b0e567ae5fc6`	78	*Aspergillus*

These three all appears on the edit-graph separated from a red node (database entry) by a dashed or dotted line indicating a 2bp or 3bp edit away.

Using an online NCBI BLAST search didn't pin any of these down to species level, but they do all seem to be fungi. Again, quite a few *Fusarium* matches which could be alternative ITS2 sequences in the genomes but not in the curated reference set. Likewise the *Aspergillus* like sequence could be from the *Aspergillus flavus* in the mock community.

i.e. None of the ITS2 sequences in amplicon library two are clear cut external contamination.

4.5 Great Lakes Mock Community 16S

This example is based on:

> Klymus *et al.* (2017) Environmental DNA (eDNA) metabarcoding assays to detect invasive invertebrate species in the Great Lakes. https://doi.org/10.1371/journal.pone.0177643

Our main focus is 5 mock communities of 11 marine species in different ratios (Table 2). The target amplicon copy number varies from trace level (14 reads) to high copy number (9090 reads), making this an interesting example to examine THAPBI PICT's minimum read abundance setting.

Two different sets primers were used targeting overlapping regions of the mtDNA 16S RNA marker gene, named MOL16S and SPH16S, which were sequenced separately (and not as in some of the other examples pooled together for the Illumina sequencing).

The full dataset includes aquarium and river environmental samples too, but public sequence databases lack many of the sequences detected.

4.5.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/great_lakes/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

Subdirectories `MOL16S/` and `SPH16S/` are used for the different amplicons (with different primer settings and reference databases).

FASTQ data

File `PRJNA379165.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata. Derived file `metadata.tsv` contains report-ready metadata about the samples (see below).

Script `setup.sh` will download the raw FASTQ files for Klymus *et al.* (2017) from https://www.ebi.ac.uk/ena/data/view/PRJNA379165

It will download 36 raw FASTQ files (18 pairs), taking 1.8GB on disk.

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/
$ md5sum -c MD5SUM.txt
...
$ cd ../
```

There is no need to decompress the files.

Amplicon primers & reference sequences

The `MOL16S` amplicon targeted a short fragment of the mtDNA 16S RNA gene using degenerate primer pair MOL16S_F/MOL16S_R (`RRWRGACRAGAAGACCCT` and `ARTCCAACATCGAGGT`).

The `SPH16S` amplicon targeted sphaeriid mussel species where it amplified an overlapping slightly downstream region of the mtDNA 16S RNA gene using non-degenerate primers SPH16S_F/SPH16S_R (`TAGGGGAAGGTATGAATGGTTTG` and `ACATCGAGGTCGCAACC`).

This means we need to run THAPBI PICT twice (once for each primer pair, against a different marker database each time).

Metadata

The provided file `metadata.tsv` is based on metadata in the ENA, split into separate columns for reporting. It has five columns:

1. Run accession, e.g. "SRR5534972"
2. Library name, e.g. "SC3PRO2"
3. Sample title, e.g. "Mock Community 2 MOL16S with Fish Block Primer"
4. Marker, "MOL16S" or "SPH16S"
5. Group, "Mock Community", "Aquarium", "River" or "Control"

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 1 -c 4,5,3,2
```

Argument `-t metadata.tsv` says to use this file for the metadata.

Argument `-c 4,5,3,2` says which columns to display and sort by. This means Marker, Group, Sample Title, Library name. This splits up the samples first by the expected marker, and then the group.

Argument `-x 1` the filename stems can be found in that column one.

Other files

Files `MOL16S.fasta` and `SPH16S.fasta` are for building reference databases. These were constructed from the accessions in the paper listed in Table 1, Table 8, Supplementary Table 1, Supplementary Table 3, and some additional accessions for the mock community. The sequences were primer trimmed using `cutadapt` (requiring both the left and right primer to be present), and the description given cut to just species level (discarding strain or isolate information).

4.5.2 Minimum Abundance Threshold

THAPBI PICT has a default minimum absolute abundance threshold of 100 reads per marker per sample, and 0.1% of the reads per marker per sample, before accepting any unique sequence. Background contamination and PCR noise levels will vary, so having multiple *Abundance & Negative Controls* will help set this objectively.

In this dataset there is a single negative control for the MOL16S marker, library `BIM8M` aka `SRR5534986`. However, we can also treat all the SPH16S libraries as negative controls for the MOL16S marker, and vice versa. You could do this automatically within THAPBI PICT via the `-n` or `--negctrls` command line option, but as we shall see in this example it will discard most of the data.

In order to examine an appropriate minimum abundance threshold, the `run.sh` script provided uses `-a 10 -f 0` to accept any unique sequence seen in sample at least ten times (regardless the fraction of the sample read total). This *does* allow unwanted noise though to the reports.

SPH16S

This was the more specific primer pair, expected to only amplify sphaeriid mussel species, so in general we expect less unique sequences than with the more general MOL16S primers.

Looking at some key columns in the sample report,

```
$ cut -f 1,2,4,7,9,14 summary/SPH16S.samples.onebp.tsv
<SEE TABLE BELOW>
```

Or, open `SPH16S.samples.onebp.xlsx` in Excel. Focusing on the left hand columns, you should see:

#Marker	Group	Library-name	Raw FASTQ	Cutadapt	Accepted
MOL16S	Aquarium	BIR2M	306311	2	0
MOL16S	Aquarium	BIR6M	291954	14	0
MOL16S	Control	BIM8M	2433	0	0
MOL16S	Mock Community	SC3PRO1	689712	17	0
MOL16S	Mock Community	SC3PRO2	405048	0	0
MOL16S	Mock Community	SC3PRO3	402219	16	0
MOL16S	Mock Community	NFSC3PRO3	349590	33	10
MOL16S	Mock Community	SC3PRO4	671241	6	0
MOL16S	Mock Community	NFSC3PRO4	420015	7	0
MOL16S	Mock Community	SC3PRO5	480606	13	0
MOL16S	River	BIM6M	821849	0	0
MOL16S	River	BIM2M	1119271	0	0
MOL16S	River	BIM4M	709472	40	19
SPH16S	Aquarium	BIR2S	498926	251148	209402
SPH16S	Aquarium	BIR6S	240360	226012	191456
SPH16S	Mock Community	SPSC3PRO1	425271	317960	224689
SPH16S	Mock Community	SPSC3PRO2	341476	282516	204249
SPH16S	Mock Community	SPSC3PRO4	410780	303957	197507

Things to note:

- In the "Raw FASTQ" column, the control has far fewer raw reads (good).
- The "Cutadapt" column shows reads after SPH16S primer trimming. There are hundreds of thousands for the final five samples amplified with these primers (good). The first 13 samples were amplified with the MOL16S primers, but still have low levels of sequences matching the SPH16S primers (bad).
- The "Read count" column is after applying the minimum abundance threshold (here 10). Two negative controls still have reads, lifting the threshold to 20 or more would fix this. These are *Sphaerium simile* in mock community `NFSC3PRO3`, and an unknown in river sample `BIM4M`.

So, using the MOL16S samples as negative controls suggests that for the SPH16S the default minimum abundance threshold is perhaps overly harsh - but using *at least* 20 would be wise.

MOL16S

We'll initially looking at the same key columns in the sample report,

```
$ cut -f 1,2,4,7,9,14 summary/MOL16S.samples.onebp.tsv
<SEE TABLE BELOW>
```

Or, open `MOL16S.samples.onebp.xlsx` in Excel. Focusing on the left hand columns, you should see:

#Marker	Group	Library-name	Raw FASTQ	Cutadapt	Accepted
MOL16S	Aquarium	BIR2M	306311	297657	256386
MOL16S	Aquarium	BIR6M	291954	286427	256470
MOL16S	Control	BIM8M	2433	1014	551
MOL16S	Mock Community	SC3PRO1	689712	656661	550293
MOL16S	Mock Community	SC3PRO2	405048	377026	297877
MOL16S	Mock Community	SC3PRO3	402219	380347	304626
MOL16S	Mock Community	NFSC3PRO3	349590	328956	262963
MOL16S	Mock Community	SC3PRO4	671241	628644	494257
MOL16S	Mock Community	NFSC3PRO4	420015	364233	262726
MOL16S	Mock Community	SC3PRO5	480606	458896	383865
MOL16S	River	BIM6M	821849	799349	703578
MOL16S	River	BIM2M	1119271	954787	823782
MOL16S	River	BIM4M	709472	367539	317363
SPH16S	Aquarium	BIR2S	498926	25	0
SPH16S	Aquarium	BIR6S	240360	27	0
SPH16S	Mock Community	SPSC3PRO1	425271	35	0
SPH16S	Mock Community	SPSC3PRO2	341476	168	27
SPH16S	Mock Community	SPSC3PRO4	410780	420	108

Looking at the same points, I see two problems:

- The control sample BIM8M (SRR5534986) had almost a thousand unwanted MOL16S matches, reduced to 551 with a minimum abundance threshold of 10.
- All the SPH16S mock community samples have unwanted MOS16S matches, the worst case being SPSC3PRO4 (SRR5534980) with over four hundred reads reduced to 108 with the minimum abundance threshold of 10.

To see exactly what is in these two problematic samples, we can turn to the read report `summary/MOL16S.reads.onebp.xlsx` in Excel, or the TSV version at the command line (using grep to drop the rows ending with a zero count):

```
$ cut -f 1,2,3,7,10 summary/MOL16S.reads.onebp.tsv | grep -v "[[:space:]]0$"
#                                                   Marker          ␣
↪MOL16S
#                                                   Group           ␣
↪Control
#                                                   Sample          ␣
↪Blank MOL16S
#                                                   Library-name    BIM8M
#                                                   Raw FASTQ       2433
#                                                   Flash           1963
#                                                   Cutadapt        1014
#                                                   Threshold pool  raw_
↪data
#                                                   Threshold       10
```

(continues on next page)

(continued from previous page)

```
#                                                                 Control          ␣
↪Sample
#                                                                 Singletons       258
#                                                                 Accepted         551
#                                                                 Unique           4
#Marker       MD5                                onebp-predictions  Total-abundance ␣
↪SRR5534986
MAX or TOTAL  -                                  -                  4914872          478
MOL16S        20c0669e4c6f8436c9d42736df727c83   Sphaerium simile   152924           478
MOL16S        e1d838b4f39bffe88d8c0e79b52700f1   Sphaerium simile   3215             13
MOL16S        778e3dace4b993135e11d450e6c559ff   Sphaerium simile   249              11
MOL16S        a36d3f7291c173c4243f22c2afbd111e   Sphaerium simile   147              49
```

The unwanted sequences in the control sample are dominated by a single sequence (and three variants of it), which were matched to *Sphaerium simile*.

This is consistent with the original author's analysis - although our pipeline has produced higher read counts:

> Finally, our water blank sample had 71 reads, eight of those being singletons with the remaining belonging to *Sphaerium striatinum* (Table 9), likely due to amplicon contamination in the lab.

What about the other problematic sample? Again, you can find this in the Excel read report, or at the command line:

```
$ cut -f 1,2,7,25 summary/MOL16S.reads.onebp.tsv | grep -v "[[:space:]]0$"
#                                                 Marker           SPH16S
#                                                 Group            Mock Community
#                                                 Sample           Mock Community 4 SPH16S
#                                                 Library-name     SPSC3PRO4
#                                                 Raw FASTQ        410780
#                                                 Flash            375539
#                                                 Cutadapt         420
#                                                 Threshold pool   raw_data
#                                                 Threshold        10
#                                                 Control          Sample
#                                                 Singletons       272
#                                                 Accepted         108
#                                                 Unique           3
#Marker       MD5                                Total-abundance  SRR5534980
MAX or TOTAL  -                                  4914872          46
MOL16S        ecdaa082b70f5e268f76128693531760   269109           45
MOL16S        98dc259e48de3e258cb93a34c38a9484   120026           17
MOL16S        20c0669e4c6f8436c9d42736df727c83   152924           46
```

The species names are too long to include in the above, listing them directly:

```
$ grep -E
↪"(MD5|20c0669e4c6f8436c9d42736df727c83|ecdaa082b70f5e268f76128693531760|98dc259e48de3e258cb93a34c38a9
↪" \
  summary/MOL16S.reads.onebp.tsv | cut -f 2,3
<SEE TABLE BELOW>
```

Giving:

MD5	onebp-predictions
ecdaa082b70f5e268f76128693531760	Dreissena bugensis;Dreissena rostriformis
98dc259e48de3e258cb93a34c38a9484	Dreissena polymorpha
20c0669e4c6f8436c9d42736df727c83	Sphaerium simile

The unwanted mock community sample content is split between *Sphaerium* and *Dreissena*, and suggest using a minimum threshold of perhaps 50 reads?

Minimum threshold

Clearly using a minimum abundance threshold of 10 is too low, and it should be increased to at perhaps 50 based on this. However, we have one exceptional sequence present at almost 500 copies. Setting the minimum that high seems excessive - but perhaps the THAPBI PICT default of 100 is more reasonable?

4.5.3 Presence and absence

This example includes mock communities which are a controlled setup where we know what the classifier ought ideally to report for every sample - and all their expected marker sequences are in the classification database.

The `thapbi_pict assess` command run via example script `run.sh` uses a configuration file with all the mock community species for MOL16S, and the three sphaeriid mussel species for SPH16S - regardless of the target copy number in the mixture (see Klymus *et al.* (2017) Table 2), or presence/absence of the fish block.

Of course, just as in the original author's analysis, not everything we think was present is detected. And *vice versa*, we see some things which are not classified.

SPH16S

This was the more specific primer pair, expected to only amplify sphaeriid mussel species, so in general we expect less unique sequences than with the more general MOL16S primers.

Only three members of the mock community should match. Looking at the `summary/SPH16S.assess.onebp.tsv` output file in Excel or at the command line, when run at a minimum abundance threshold of 10, these are the key numbers:

```
$ cut -f 1-5,9,11 summary/SPH16S.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	9	5	0	656	0.78	0.357
Pisidium compressum	3	0	0	7	1.00	0.000
Sphaerium corneum	3	0	0	7	1.00	0.000
Sphaerium nucleus	0	3	0	7	0.00	1.000
Sphaerium simile	3	1	0	6	0.86	0.250
Sphaerium striatinum	0	1	0	9	0.00	1.000
OTHER 62 SPECIES IN DB	0	0	0	620	0.00	0.000

No false negatives (but we have set the threshold very low), but 5 false positives: Three cases of *Sphaerium nucleus*, and one each of *S. simile* and *S. striatinum*.

The *S. nucleus* matches are simply down to an ambiguous sequence in the database from both this and expected species *S. corneum*. See also the output from `thapbi_pict conflicts -d SPH16S.sqlite` which can report this.

The *S. striatinum* prediction came from `SPSC3PRO1` aka `SRR5534978`, and is down to several sequences one base pair away the expected *S. simile* reference, but also one base pair away from an *S. striatinum* database entry.

We already discussed the trace level of 10 reads for *Sphaerium simile* in mock community sample `NFSC3PRO3` using the SOL16S primers. As suggested, raising the minimum abundance threshold to at least 20 reads would solve this, but the other false positives here are limitations of the reference set.

MOL16S

Looking at the `summary/MOL16S.assess.onebp.tsv` output file in Excel or at the command line, when run at a minimum abundance threshold of 10, these are the key numbers:

```
$ cut -f 1-5,9,11 summary/MOL16S.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	74	23	3	1220	0.85	0.260
Cipangopaludina chinensis	7	0	0	4	1.00	0.000
Corbicula fluminea	0	1	0	10	0.00	1.000
Dreissena bugensis	0	8	0	3	0.00	1.000
Dreissena polymorpha	7	1	0	3	0.93	0.125
Dreissena rostriformis	7	1	0	3	0.93	0.125
Gillia altilis	7	0	0	4	1.00	0.000
Melanoides tuberculata	7	0	0	4	1.00	0.000
Mytilopsis leucophaeata	7	0	0	4	1.00	0.000
Pisidium compressum	7	0	0	4	1.00	0.000
Potamopyrgus antipodarum	7	0	0	4	1.00	0.000
Sander vitreus	4	0	3	4	0.73	0.429
Sphaerium corneum	7	1	0	3	0.93	0.125
Sphaerium nucleus	0	8	0	3	0.00	1.000
Sphaerium simile	7	2	0	2	0.88	0.222
Sphaerium striatinum	0	1	0	10	0.00	1.000
OTHER 105 SPECIES IN DB	0	0	0	1155	0.00	0.000

This time we do have false negatives - three of the seven samples are missing *Sander vitreus*. Two of these are from Community 3 where this is intended to be at only 14 copies, the third was `SC3PRO2` aka `SRR5534972` for Mock Community 2 MOL16S with Fish Block Primer, with a target abundance of 72 copies. Here the fish block worked.

Again we have lots of false positives, mostly sister species which reflects limitations of the reference set.

The exception is *Corbicula fluminea*. Referring to the sample summary report `MOL16S.samples.onebp.xlsx`, this is from `SC3PRO1` aka `SRR5534973`, and at low abundance. This species was present in the aquaria sample sediment, but as discussed in the paper did not amplify from there - so cross-contamination seem less likely.

Unknowns

Looking at `SPH16S.samples.onebp.xlsx` and `MOL16S.samples.onebp.xlsx` even our controls have unknown reads. To study these, next I'd look at the edit-graphs.

4.5.4 Edit Graphs

The sequence *Edit Graph* is very useful for understanding what came off the sequencer - although you may need to play with the thresholds to find a sweet spot for hiding the noise. Using `run.sh` calls the pipeline with a minimum abundance 10, which would give large noisy edit graphs. Instead, we build them using a minimum abundance of 100, giving files `SPH16S.edit-graph.a100.xgmml` and `MOL16S.edit-graph.a100.xgmml`, and additional graphs for the mock community samples alone.

My main conclusion from the figures below is that the THAPBI PICT default `onebp` classifier is reasonable for these mock communities markers. However, the MOL16S database needs considerable expansion for use on the environmental samples. Perhaps updating this example in 5 years time there will be enough published markers to assign species to all the unknowns here?

SPH16S

First SPH16S, where there are just the three samples for the mock communities. Each is expected to have three species *Sphaerium simile*, *Sphaerium corneum* and *Pisidium compressum* only. With a minimum abundance threshold of 100, we get three nice clear graph components, and a few single nodes:

Next, using all the samples but again a sample level minimum abundance 100:

Very little change except the addition of a fourth cluster, some base pairs away from the *Sphaerium simile* component and centred on this sequence:

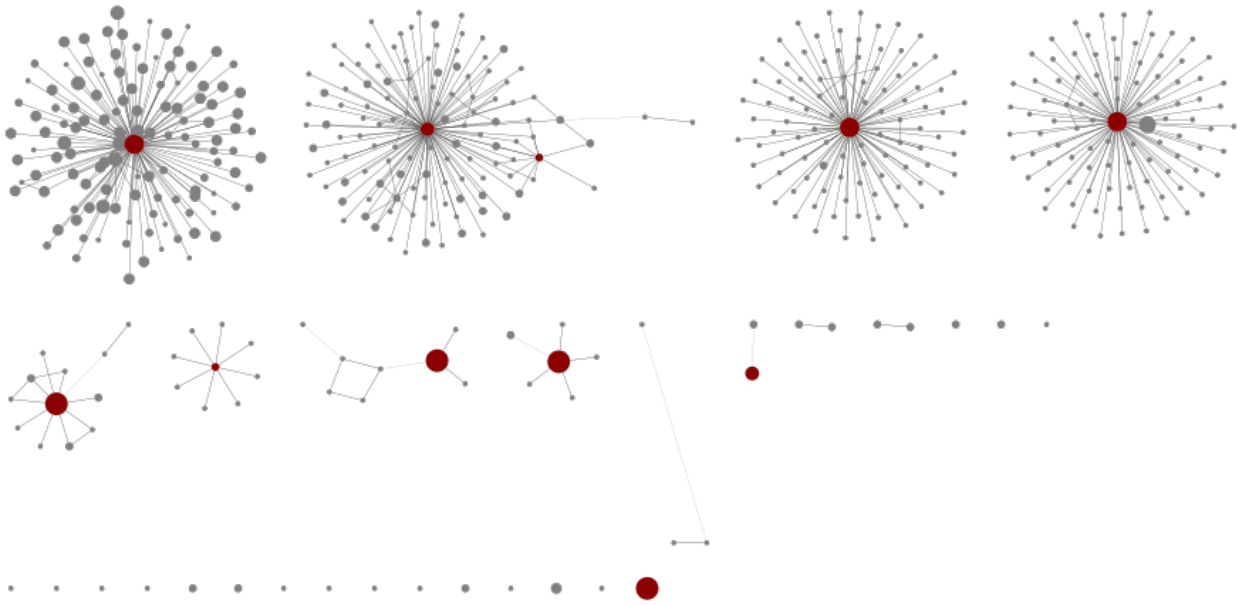
```
>79b63a2ef96b839ae3263369f8d390b9
ACGTGGAAAAAACTGTCTCTTTTGTATAAAAAGAAGTTTATTTTTAAGTGAAAAAGCTTAAATGTTTATAAAAGACGAGA
AGACCCTATCGAACTTAAATTATTTGTTTAAATTTTTAAATAAAAAAAAAGTTTAGTTGGGGAAACTTAAAGTAAAAAGTA
ACGCTTTATTTTTTTGTCAGGAGCCTGTAGTATGGAAAAATGAAAAAGTTACCGTAGGGATAACAGCGCTTTCTTCTCTG
AGAGGACTAATTAAAGAGTT
```

This is likely another *Sphaerium* species, NCBI BLAST suggests *Sphaerium striatinum* - with AF152045.1 just two base pair away. This is in our reference database file `SPH16S.fasta`:

```
>AF152045.1 Sphaerium striatinum
ACGTGGAAAAAACTGTCTCTTTTGTATAAAAAGAAGTTTATTTTTAAGTGAAAAAGCTTAAATGTTTATAAAAGACGAGA
AGACCCTATCGAACTTAAATTATTTGTTTAAATTTTTAAATAAAAAAAAAGTTTAGTTGGGGAAACTTAAAGTAAAAATTA
ACGCTTTATTTTTTTGTCAGGAGCCTGTACTATGGAAAAATGAAAAAGTTACCGTAGGGATAACAGCGCTTTCTTCTCTG
AGAGGACTAATTAAAGAGTT
```

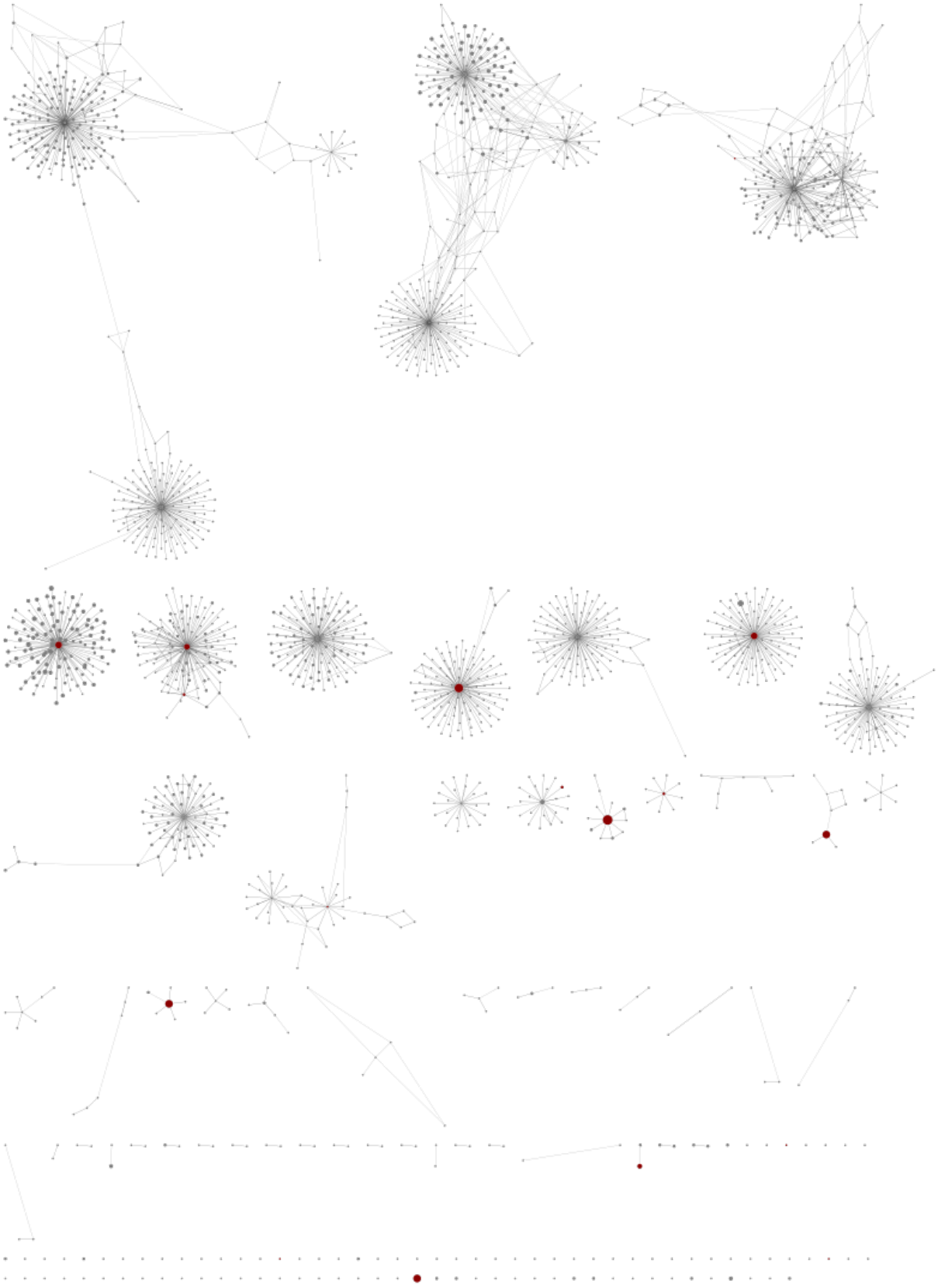
MOL16S

For MOL16S, starting with an edit graph of just the seven MOL16S samples, and a minimum abundance threshold of 100, we see:



Four large components representing species with lots of variants, with red central nodes in our database. Other less diverse graph components for the remaining species, and a selection of isolated unknowns.

Next, using all the samples but again a sample level minimum abundance 100:



Suddenly we see dozens of new components, most of which have no references (coloured nodes) representing likely unknown species.

Conclusion

I will close by quoting the end of Klymus *et al.* (2017):

> The present study further demonstrates that metabarcoding data are only as good as the sequence and taxonomic information provided on genetic databases. Increased collaboration among taxonomists and molecular systematists is required in order to gain maximum benefits of this developing tool.

I agree - these markers seem to work, but there are still too many unknown sequences.

4.6 Bat Mock Community COI

This example considers mock communities of 3 bat species (in different ratios) using the COI marker, using one of the amplicon sequencing libraries from:

> Walker *et al.* (2019) A fecal sequel: Testing the limits of a genetic assay for bat species identification. https://doi.org/10.1371/journal.pone.0224969

The example highlights the importance of good database coverage with the default `onebp` classifier method.

4.6.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/fecal_sequel/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

FASTQ data

File `PRJNA574765` was download from the ENA and includes the FASTQ checksums, URLs, and the key metadata. Related file `metadata.tsv` contains report-ready metadata about the samples (see below).

Script `setup.sh` will download the raw FASTQ files for Walker *et al.* (2019) from https://www.ebi.ac.uk/ena/data/view/PRJNA574765

It will download 120 raw FASTQ files (60 pairs), taking about 641MB on disk

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/
$ md5sum -c MD5SUM.txt
...
$ cd ..
```

There is no need to decompress the files.

We focus on bioproject PRJNA574765 which has 60 samples and covers the mock communities. Additionally the paper describes PRJNA525109 (41 samples comparing genetic efficacy vs traditional survey techniques), and PRJNA525407 (9 samples looking at bat species assemblages in archaeological sites in Belize, with an expanded reference set).

Amplicon primers & reference sequences

The primer pair is SFF_145f (`GTHACHGCYCAYGCHTTYGTAATAAT`) and SFF_351r (`CTCCWGCRTGDGCWAGRTTTCC`).

The reference set of COI sequences is taken from Supplementary S2 in the preceding paper (which also included bioproject PRJNA325503 with 9 samples):

> Walker *et al.* (2016) Species From Feces: Order-Wide Identification of Chiroptera From Guano and Other Non-Invasive Genetic Samples. https://doi.org/10.1371/journal.pone.0162342

File `COI_430_bats.fasta` of pre-trimmed bat COI markers is generated by `setup.sh` by downloading the FASTA file from Walker *et al.* (2016) Supplementary S2, with underscores replaced with spaces in the record names.

Provided file `observed_3_bats.fasta` contains alternative COI markers observed in at least 10 samples, and their assumed species source. This is for discussing the effect of the database.

Metadata

The provided file `metadata.tsv` is based on `PRJNA574765` but breaks up the sample name into separate columns:

1. Accession, assigned by the public archive, e.g. "SRR10198789"
2. Rare, which of the 3 species is at low abundance, "COTO", "EPFU" or "TABR".
3. Ratio, either "1:64" (rare) or "1:192" (very rare)
4. Replicate, "01" to "10" (leading zero for alphabetical sorting)

The four letter appreviations are *Corynorhinus townsendii* (COTO), *Eptesicus fuscus* (EPFU) and *Tadarida brasiliensis* (TABR).

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 1 -c 2,3,4
```

Argument `-t metadata.tsv` says to use this file for the metadata.

The `-x 1` argument indicates the filename stem can be found in column 1, Accession.

Argument `-c 2,3,4` says which columns to display and sort by (do not include the indexed column again). i.e. Rare species, ratio, replicate.

We have not given a `-g` argument to assign colour bands in the Excel reports, so it will default to the first column in `-c`, meaning we get three coloured bands for "COTO", "EPFU" and "TABR".

Other files

File `mock_community.known.tsv` describes the three species of bats expected in the mock communities (which use different ratios).

4.6.2 Database of 430 bats

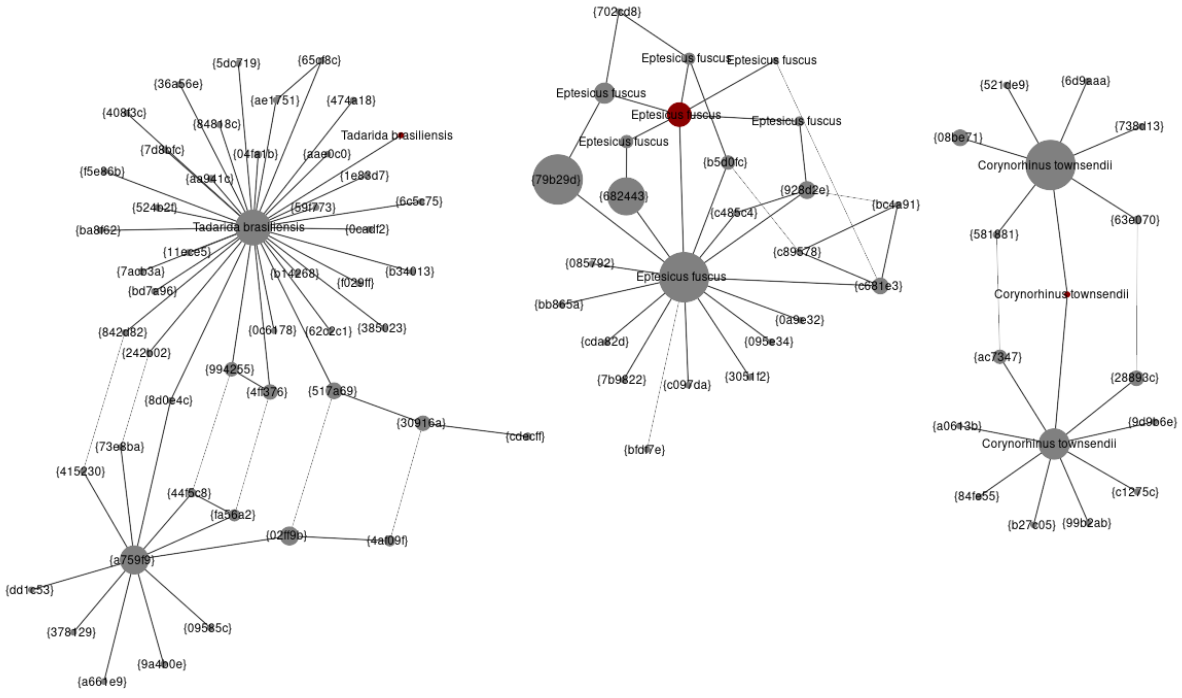
The mock communities of 3 bats in Walker *et al.* (2019) are made up of just *Corynorhinus townsendii* (COTO), *Eptesicus fuscus* (EPFU) and *Tadarida brasiliensis* (TABR). Following their analysis, we will use the same 430 bat species for our reference database, available as a FASTA file of COI trimmed markers as Supplementary S2 to the earlier paper Walker *et al.* (2016).

Running `setup.sh` will download that FASTA file as `COI_430_bats.fasta`, and we can load this into a new THAPBI PICT database using:

```
$ rm -rf COI_430_bats.sqlite  # delete any pre-existing DB
$ thapbi_pict import -k COI \
  --left GTHACHGCYCAYGCHTTYGTAATAAT --right CTCCWGCRTGDGCWAGRTTTCC \
  -d COI_430_bats.sqlite -i COI_430_bats.fasta -x
File COI_430_bats.fasta had 430 sequences, of which 430 accepted.
Of 430 potential entries, loaded 430 entries, 0 failed parsing.
```

Here we have named the new marker COI, and recorded the SFF_145f/SFF_351r primer pair. Calling `run.sh` will first run the pipeline using this COI database and primers, and the metadata as described earlier. This will make an edit-graph named `summary/430_bats.COI.edit-graph.xgmml` which you can open in Cytoscape. This contains three main connected components for the three expected species, and a smattering of singletons and other tiny clusters. The `-k` or `--marker` option was used to force inclusion of the database entries (even if not seen in the samples).

Importantly, only the *Eptesicus fuscus* cluster includes a red node from the database which is also in the samples. i.e. None of the sequence data from *Corynorhinus townsendii* or *Tadarida brasiliensis* perfectly matches the given reference species sequence. The three main clusters are shown below:



All three clusters have species labelled nodes. Starting on the left, we have *Tadarida brasiliensis* where the reference is a one base pair edit away from the dominant variant (seen in 27 samples). In the middle we have *Eptesicus fuscus* where while the reference sequence was seen, once again it is not a dominant variant (two variants were seen in 40 samples). Finally, on the right for *Corynorhinus townsendii* the reference is a one base pair edit away from the two dominant variants (seen in 40 and 23 samples).

This is a severe handicap for the default `onebp` classifier which looks for identical matches or up to a single base pair different. We can either switch to a fuzzier classifier (like the `blast` based classifier), or look at filling in the database.

One option would be to add a (curated subset of) published sequences from the NCBI. At the time of writing while that helps, there are still gaps here. What the example in `run.sh` does is add all the sequences observed in at least 10 samples to the database with the presumed species. This is similar to how the THAPBI PICT default database of *Phytophthora* ITS1 contains actually observed variants from single species positive controls. It ceases to be an entirely fair assessment, but comparing the Excel reports from the two database the improvement is quite dramatic.

Looking at `summary/430_bats.COI.samples.onebp.xlsx` (430 references only) compared to `summary/ext_bats.COI.samples.onebp.xlsx` (with an extra 14 sequences added), the *Tadarida brasiliensis* detection improves markedly (although is still spotted in only two of the 20 replicates where it is the rare species - consistent with the published analysis and put down to primer preference), and also there are far less unknown reads reported.

4.6.3 Pooling

This is a nice example to show the pooling script included with THAPBI PICT, here pooling on the first two columns of the sample report:

```
$ ../../scripts/pooling.py -i summary/430_bats.COI.samples.onebp.tsv -c 1,2
<SEE TABLE BELOW>
```

You can specify an output stem like `-o pooled` and get `pooled.tsv` and matching `pooled.xlsx` files, but by default the plain text table is printed to the terminal:

Rare	Ratio	Samples-sequenced	Corynorhinus townsendii	Eptesicus fuscus	Tadarida brasiliensis	Unknown
COTO	1:192	10	58948	99888	82587	19059
COTO	1:64	10	45632	51977	0	148446
EPFU	1:192	10	99840	9668	103545	21191
EPFU	1:64	10	91018	52574	21507	65809
TABR	1:192	10	149636	73958	1563	52279
TABR	1:64	10	128019	106581	773	50833

As discussed earlier, where *Corynorhinus townsendii* (COTO) is the rare species at a 1:64 ratio there is no *Tadarida brasiliensis* matched with the initial database, but it is found with the extended database:

```
$ ../../scripts/pooling.py -i summary/ext_bats.COI.samples.onebp.tsv  -c 1,2
<SEE TABLE BELOW>
```

Again, shown as a table:

Rare	Ratio	Samples-sequenced	Corynorhinus townsendii	Eptesicus fuscus	Tadarida brasiliensis	Unknown
COTO	1:192	10	61727	100185	92815	5755
COTO	1:64	10	70121	68495	101333	6106
EPFU	1:192	10	100822	9668	108264	15490
EPFU	1:64	10	91242	68322	67690	3654
TABR	1:192	10	154907	98791	1563	22175
TABR	1:64	10	133876	140456	773	11101

One of the options in this script is `-b` or `--boolean` for a yes/no summary rather than showing the sum of the reads:

```
$ ../../scripts/pooling.py -i summary/ext_bats.COI.samples.onebp.tsv  -c 1,2 -b
<SEE TABLE BELOW>
```

All three species (and unknowns) are found in at least one of the 10 samples sequenced in each of the six groups:

Rare	Ratio	Samples-sequenced	Corynorhinus townsendii	Eptesicus fuscus	Tadarida brasiliensis	Unknown
COTO	1:192	10	Y	Y	Y	Y
COTO	1:64	10	Y	Y	Y	Y
EPFU	1:192	10	Y	Y	Y	Y
EPFU	1:64	10	Y	Y	Y	Y
TABR	1:192	10	Y	Y	Y	Y
TABR	1:64	10	Y	Y	Y	Y

In the Excel output the species labels are rotated 90 degrees allowing a very compact display.

4.7 Synthetic controls with fungal ITS2

Here we consider some environmental fungi, mock communities, and a synthetic spike-in control using some of the Illumina data from the following paper:

> Palmer *et al.* (2018) Non-biological synthetic spike-in controls and the AMPtk software pipeline improve mycobiome data. https://doi.org/10.7717/peerj.4925 https://www.ebi.ac.uk/ena/data/view/PRJNA305924

The mock communities have known composition, while the synthetic control has a mix of 12 artificial sequences which can be easily distinguished from the biological ITS2 sequences.

4.7.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/synthetic_mycobiome/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

FASTQ data

File `PRJNA305924.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata (not just for the files we will be using, but additional Illumina MiSeq runs, and Ion Torrent data too).

Script `setup.sh` will download the raw FASTQ files for two of the Illumina MiSeq runs described in Palmer *et al.* (2018) from https://www.ebi.ac.uk/ena/data/view/PRJNA305924

It will download 42 raw FASTQ files (21 pairs), taking about 4.8 GB on disk.

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd ..
$ md5sum -c MD5SUM.txt
...
$ cd ..
```

There is no need to decompress the files.

Amplicon primers & reference sequences

A region of ITS2 was amplified using the fITS7/ITS4 primer pair (`GTGARTCATCGAATCTTTG` and `TCCTCCGCTTATTGATATGC`) with an average product length of 264bp using public fungal sequences.

The file `references.fasta` we provide is based on `amptk_mock2.fa` and `amptk_mock3.fa` from the authors' GitHub repository <https://github.com/nextgenusfs/amptk/tree/master/amptk/DB>, but formatted suitable for direct import into our tool with primer-trimmed sequences.

Additional file `environment.fasta` contains selected close matches to sequences from the environmental samples in the NCBI found with BLASTN against the NT database.

Metadata

File `metadata.tsv` is based on the ENA metadata and the paper text. It has four columns:

1. run_accession, assigned by the public archive, e.g. "SRR7109326"
2. library_name, with sequencing run as a prefix, e.g. "m6-stds" or "m6-301-1"
3. sample_alias, as used in the paper, e.g. "BioMockStds" or "301-1"
4. group, human readable sample type, e.g. "Biological Mock" or "Environment"
5. read_count, the number of read pairs in the FASTQ files

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 1 -c 3,4
```

Argument `-t metadata.tsv` says to use this file for the metadata.

Argument `-c 3,4` says which columns to display and sort by. This means sample alias, then group.

Argument `-x 1` (default, so not needed) indicates the filename stem can be found in column 1, run accession.

Other files

Provided files `BioMockStds.known.tsv`, `BioMock.known.tsv`, and `SynMock.known.tsv` list the expected 25 species, 22 species, and 12 synthetic controls expected in the mock samples. Folder `expected/` is created linking accession names to the appropriate species for assessing the classifier performance.

Sub-folder `intermediate/ITS2/` is used for intermediate files, in general there is a sub-folder for each primer-pair.

4.7.2 Minimum Abundance Threshold

With less samples multiplexed per sample than our own work (which guided the default settings), these samples were sequenced at much higher depth:

```
$ cut -f 1,2,5 metadata.tsv
<SEE TABLE BELOW>
```

As a table:

run_accession	library_name	read_count
SRR7109326	m6-stds	817764
SRR7109327	m6-301-1	890561
SRR7109328	m6-mock3-32000b	943839
SRR7109329	m6-766-1	840068
SRR7109330	m6-744-2	704173
SRR7109331	m6-500-1	911793
SRR7109341	m6-712-2	872265
SRR7109342	m6-500-2	879762
SRR7109343	m6-757-1	903886
SRR7109344	m6-757-2	1210627
SRR7109345	m6-mock3-16000	922440
SRR7109406	m6-712-1	897159
SRR7109408	m6-744-1	778090
SRR7109409	m6-mock3-32000a	1125275
SRR7109411	m6-766-2	785776
SRR7109412	m6-755-1	957067
SRR7109414	m6-736-1	998817
SRR7109415	m6-301-2	1181567
SRR7109417	m6-755-2	1071829
SRR7109418	m6-736-2	919363
SRR7109420	m6-SynMock	1299238

The defaults are an absolute abundance threshold of 100, and a fractional threshold of 0.1% (i.e. `-a 100 -f 0.001`). After merging overlapping reads and primer matching we could expect over 650,000 reads per m6 sample, giving a threshold over 650 reads.

So, with this coverage the default fractional abundance threshold of 0.1% (i.e. `-f 0.001`) makes the default absolute abundance threshold of 100 (i.e. `-a 100`) redundant. However, on this dataset our defaults are quite cautious, and control samples can help set thresholds objectively.

In this dataset there is a single synthetic control for m6 sequencing run, library `SynMock` aka `SRR7109420`. We can tell THAPBI PICT at the command line to use this to set the fractional abundance threshold via `-y` or `--synctrls`, and/or set the absolute abundance threshold via `-n` or `--negctrls` (with a list of control file names). It turns out however that with the default thresholds the control is clean (no unwanted non-synthetic ITS2 reads).

Using the defaults

The first step in `run.sh` is to run the pipeline with the default abundance thresholds (stricter than the alternatives analyses below), giving just a few hundred unique ITS2 sequences:

```
$ grep -c "^ITS2" summary/defaults.ITS2.tally.tsv
360
$ grep -c "^ITS2" summary/defaults.ITS2.reads.1s5g.tsv
360
```

Look at `summary/defaults.ITS2.samples.1s5g.xlsx` or working at the command line with the TSV file:

```
$ cut -f 1,7,9,11-12,14-15 summary/defaults.ITS2.samples.1s5g.tsv
<SEE TABLE BELOW>
```

As a table:

#sample_alias	Cutadapt	Threshold	Max non-spike	Max spike-in	Accepted	Unique
301-1	807956	808	348111	0	528957	38
301-2	1108129	1109	457440	0	778850	31
500-1	819468	820	289229	0	516474	30
500-2	813470	814	214155	0	529967	34
712-1	820146	821	131937	0	533310	56
712-2	796363	797	299240	0	520290	34
736-1	943427	944	349965	0	669563	36
736-2	854919	855	282132	0	609025	25
744-1	706659	707	358089	0	493209	20
744-2	651528	652	136471	0	452421	36
755-1	887650	888	462493	0	616322	27
755-2	982087	983	589120	0	669602	17
757-1	835431	836	281533	0	578198	35
757-2	1099959	1100	224635	0	742540	28
766-1	792260	793	526535	0	583643	16
766-2	711176	712	251097	0	469397	26
BioMock	866253	867	56120	0	591947	23
BioMock	846519	847	65686	0	585715	23
BioMock	1023231	1024	84748	0	698170	22
BioMockStds	736334	737	35300	0	521693	26
SynMock	1199806	1200	0	103014	862950	18

The `SynMock` control is clean, no non-spike-in reads passed the default abundance thresholds.

So, there is scope to lower the default thresholds - but how low? We will start by reproducing the Illumina part of Figure 6, which was based on the m6 MiSeq sequencing run. This figure explores tag-switching in the demultiplexing, and in the authors' analysis goes as low as 5 reads.

Excluding only singletons

The `run.sh` example continues by running the pipeline on the m6 dataset with `-f 0 -a 2` to accept everything except singletons (sequences which are only seen once in a sample; including them gives about ten times as many unique sequences which slows everything down). Also, this analysis does *not* use the synthetic control to raise the threshold on the rest of the samples - we want to see any low level mixing. We then can compare our sample report against Figure 6.

Looking at the unique reads in the FASTA file, tally table, or in the reads report with metadata, we have nearly 200 thousand ITS2 sequences:

```
$ grep -c "^ITS2" summary/a2.ITS2.tally.tsv
196480
$ grep -c "^ITS2" summary/a2.ITS2.reads.onebp.tsv
196480
```

Look at `summary/a2.ITS2.samples.onebp.xlsx` or working at the command line with the TSV file:

```
$ cut -f 1,5-7,11-12,14-15 summary/a2.ITS2.samples.onebp.tsv
<SEE TABLE BELOW>
```

As a table:

#sample_alias	Raw FASTQ	Flash	Cutadapt	Max non-spike	Max spike-in	Accepted	Unique
301-1	890561	812674	807956	348111	0	687950	12638
301-2	1181567	1113606	1108129	457440	0	977003	13319
500-1	911793	823392	819468	289229	0	689174	14249
500-2	879762	817277	813470	214155	0	699634	12851
712-1	897159	823034	820146	131937	0	703189	17574
712-2	872265	800475	796363	299240	0	683057	13937
736-1	998817	948348	943427	349965	15	834461	12993
736-2	919363	858915	854919	282132	0	757097	9625
744-1	778090	710762	706659	358089	0	614988	7936
744-2	704173	654661	651528	136471	0	564238	8650
755-1	957067	891942	887650	462493	15	782052	12142
755-2	1071829	987280	982087	589120	0	848793	10587
757-1	903886	839105	835431	281533	0	725057	12729
757-2	1210627	1105530	1099959	224635	0	950457	15819
766-1	840068	794475	792260	526535	0	712126	7519
766-2	785776	714894	711176	251097	0	606887	11189
BioMock	943839	872263	866253	56120	0	744007	17274
BioMock	922440	859262	846519	65686	0	733784	16676
BioMock	1125275	1047383	1023231	84748	3	884514	18416
BioMockStds	817764	740627	736334	35300	0	628576	17202
SynMock	1299238	1204532	1199806	187	103014	1043525	14234

Here `SynMock` (`SRR7109420`) is the synthetic control, and it has some non-spike-in reads present, the most abundant at 187 copies. Conversely, samples `755-1` (`SRR7109412`), `736-1` (`SRR7109414`), and one of the BioMock samples (`SRR7109409`) have trace levels of unwanted synthetic spike-in reads, the most abundant at 15, 15 and 3 copies respectively. The counts differ, but these are all samples highlighted in Figure 6 (sharing the same Illumina i7 or i5 index for multiplexing). We don't see this in the other BioMock samples, but our pipeline appears slightly more stringent.

As percentages, 187/1199806 gives 0.0156% which is nearly ten times lower than our default of 0.1%. The numbers the other way round are all even lower, 15/462496 gives 0.003%, 15/349965 gives 0.004%, and 3/1023234 gives 0.003%.

Using the synthetic control

Next the `run.sh` example uses the `SynMock` synthetic control to automatically raise the fractional abundance threshold from zero to 0.015% by including `-a 100 -f 0 -y raw_data/SRR7109420_*.fastq.gz` in the command line. This brings down the unique sequence count enough to just over three thousand, allowing use of a slower but more lenient classifier as well:

```
$ grep -c "^ITS2" summary/ctrl.ITS2.tally.tsv
3097
$ grep -c "^ITS2" summary/ctrl.ITS2.reads.1s5g.tsv
3097
```

Look at `summary/ctrl.ITS2.samples.1s5g.xlsx` or working at the command line with the TSV file:

```
$ cut -f 1,7,9,11-12,14-15 summary/ctrl.ITS2.samples.1s5g.tsv
<SEE TABLE BELOW>
```

Note we now get a threshold column showing the absolute threshold applied to each sample (using the inferred percentage), all above the absolute default of 100. You can see the total accepted read count has dropped, and the number of unique sequences accepted has dropped even more dramatically:

#sample_alias	Cutadapt	Threshold	Max non-spike	Max spike-in	Accepted	Unique
301-1	807956	126	348111	0	579502	262
301-2	1108129	173	457440	0	829870	189
500-1	819468	128	289229	0	568336	228
500-2	813470	127	214155	0	578432	215
712-1	820146	128	131937	0	569100	181
712-2	796363	125	299240	0	570488	243
736-1	943427	148	349965	0	708900	183
736-2	854919	134	282132	0	653753	220
744-1	706659	111	358089	0	540597	273
744-2	651528	102	136471	0	472785	129
755-1	887650	139	462493	0	694273	340
755-2	982087	154	589120	0	754928	338
757-1	835431	131	281533	0	610579	171
757-2	1099959	172	224635	0	781212	142
766-1	792260	124	526535	0	648524	301
766-2	711176	111	251097	0	508838	205
BioMock	866253	136	56120	0	607401	77
BioMock	846519	132	65686	0	603186	82
BioMock	1023231	160	84748	0	718660	85
BioMockStds	736334	115	35300	0	526317	48
SynMock	1199806	100	187	103014	885051	113

Note that Palmer *et al.* (2018) apply a threshold to individual sequences, but the thresholding strategy in THAPBI PICT applies the fractional threshold to all the samples (given in the same sub-folder as input, so you can separate your MiSeq runs, or your PCR plates, or just apply a global threshold).

In fact, looking at the read report `summary/ctrl.ITS2.reads.1s5g.tsv` it is clear that while this threshold may have excluded Illumina tag-switching, it has *not* excluded PCR noise - there are hundreds of low abundance sequences unique to a single sample. To address that we would have to use a considerably higher threshold, and the default 0.1% is a reasonable choice here, or apply a denoising algorithm like UNOISE.

Threshold selection

Excluding only singletons is too lenient, but how does the the synthetic control inferred threshold (0.0156%) compare to the default (0.1%)?

Here are the classifier assessment values using the lower inferred threshold which allows a lot of PCR noise:

```
$ head -n 2 summary/ctrl.ITS2.assess.1s5g.tsv
<SEE TABLE BELOW>
```

As a table:

#Species	TP	FP	FN	TN	sensitivity	specificity	precision	F1	Hamming-loss	Ad-hoc-loss
OVERALL	102	11	1	186	0.99	0.94	0.90	0.94	0.0400	0.105

Versus the stricter higher default abundance fraction which excludes most of the PCR noise:

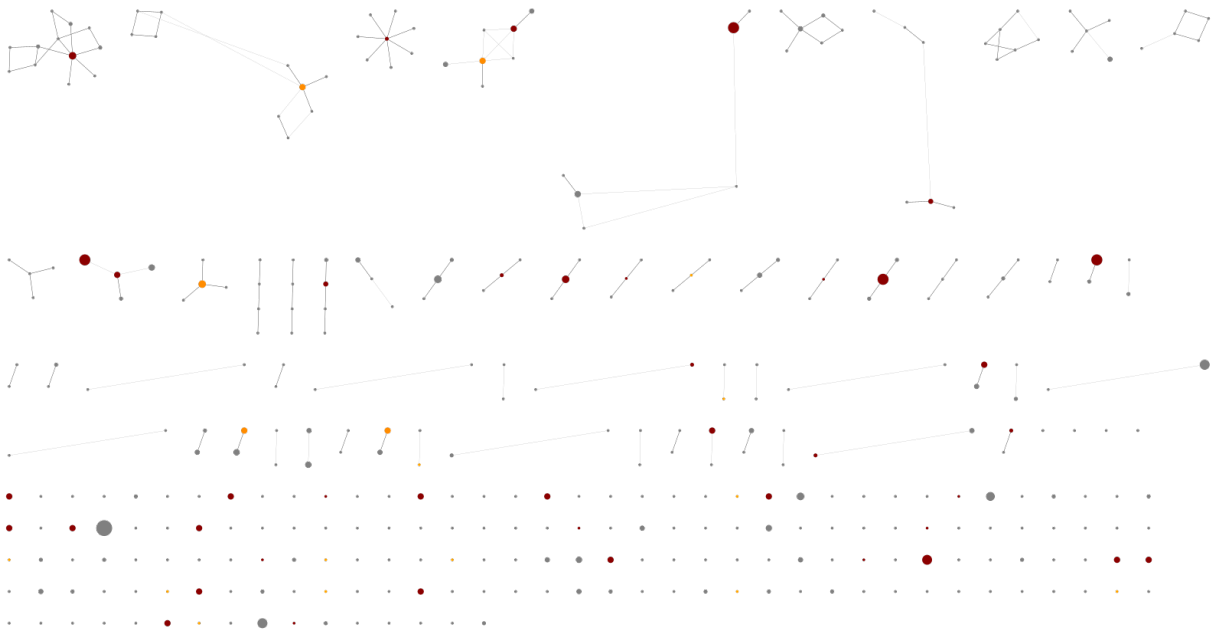
```
$ head -n 2 summary/defaults.ITS2.assess.1s5g.tsv
<SEE TABLE BELOW>
```

As a table:

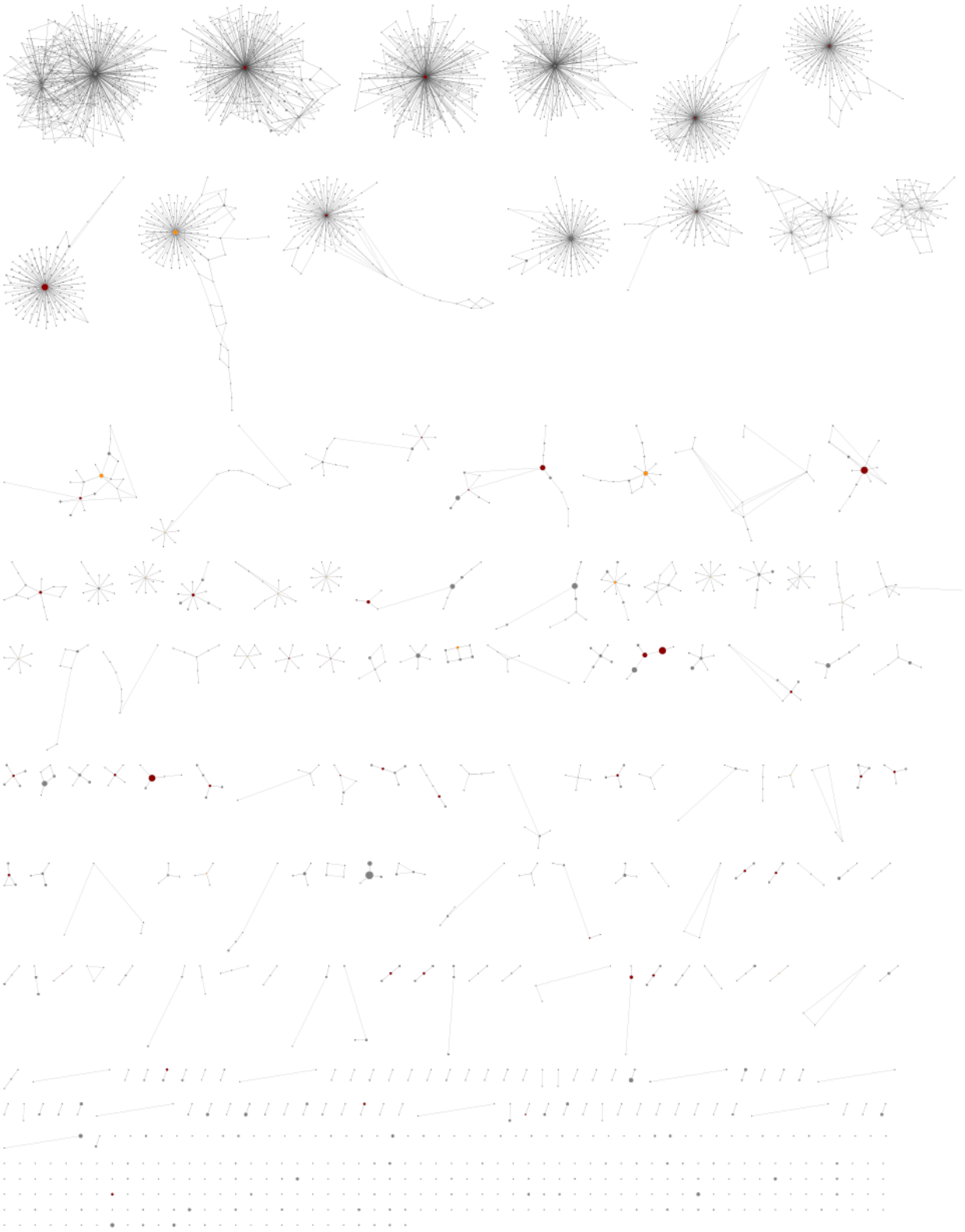
#Species	TP	FP	FN	TN	sensitivity	specificity	precision	F1	Hamming-loss	Ad-hoc-loss
OVERALL	92	8	11	189	0.89	0.96	0.92	0.91	0.0633	0.171

You could use the assessment metrics to help decide on your preferred threshold, depending on the best tradeoff for your use-case.

Personally, of the these two I would pick the higher default threshold since it appears to exclude lots of PCR noise as seen in the edit graphs. With the default 0.1% threshold:



Using the lower threshold there are roughly ten times as many ASVs. The more common ASV nodes become the centre of a halo of 1bp variants, typically each seen in a single sample, which we attribute to PCR noise:



The best choice of threshold may lie somewhere in between?

Read-correction for denoising

Read-correction is an alternative or supplement to a stringent abundance filter for removing the noise of sequence variants presumed to be PCR artefacts. Use `--denoise` as part of the pipeline or sample-tally commands to enable our implementation of the UNOISE algorithm (Edgar 2016).

Adding this to the control-driven abundance threshold example drops the total unique read count from over 3 thousand to just over 700:

```
$ grep -c "^ITS2" summary/ctrl_denoise.ITS2.tally.tsv
704
$ grep -c "^ITS2" summary/ctrl_denoise.ITS2.reads.1s5g.tsv
704
```

This gives an edit graph visually somewhere in between the examples above, with the obvious variant halos collapsed, but some of the more complex chains of variants still present.

In terms of classifier assessment on the mock community, there is no change:

```
$ head -n 2 summary/ctrl_denoise.ITS2.assess.1s5g.tsv
<SEE TABLE BELOW>
```

As a table:

#Species	TP	FP	FN	TN	sensitivity	specificity	precision	F1	Hamming-loss	Ad-hoc-loss
OVERALL	102	11	1	186	0.99	0.94	0.90	0.94	0.0400	0.105

Looking at the reports, the read counts are of course different, but also some of the reads assigned a genus-only classification have been removed via the read-correction, so the taxonomy output does not directly match up either.

4.8 Soil Nematode Mock Community

Here we consider a mock community of over 20 soil nematodes (in triplicate), sequeced separately with four markers, with no-template PCR control blanks:

> Ahmed *et al.* (2019) Metabarcoding of soil nematodes: the importance of taxonomic coverage and availability of reference sequences in choosing suitable marker(s) https://doi.org/10.3897/mbmg.3.36408 https://www.ebi.ac.uk/ena/data/view/PRJEB27581

This example requires creating a database covering the four different marker primers and known sequences (all of which ought to be properly curated).

4.8.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/soil_nematodes/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

FASTQ data

File `PRJEB27581.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata.

Script `setup.sh` will download the raw FASTQ files for Ahmed *et al.* (2019) from https://www.ebi.ac.uk/ena/data/view/PRJEB27581

It will download 32 raw FASTQ files (16 pairs), taking 12GB on disk.

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/
$ md5sum -c MD5SUM.txt
...
$ cd ../
```

There is no need to decompress the files.

Amplicon primers & reference sequences

There were four separate markers used here, as shown in the paper's Table 2 together with the shared Illumina adaptors used.

The authors do not provide copies of their reference sequence databases with the paper. Instead, files `NF1-18Sr2b.fasta`, `SSUF04-SSUR22.fasta`, `D3Af-D3Br.fasta` and `JB3-JB5GED.fasta` were based on the accessions listed in the paper and close matches in the NCBI found with BLASTN against the NT database. Note many of the species names have been reduced to just "Genus sp." in line with the mock community entries, and all the fungal entries are listed as just "Fungi".

Metadata

File `metadata.tsv` is based on the ENA metadata and the paper text. It has four columns:

1. run_accession, assigned by the public archive, e.g. "ERR2678656"
2. read_count, the number of paired reads in the raw FASTQ files.
3. sample, one of "MC1", "MC2", "MC3" for the mock communities, or "Blank"
4. marker, one of "NF1-18Sr2b", "SSUF04-SSUR22", "D3Af-D3Br" or "JB3-JB5GED"

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 1 -c 4,3
```

Argument `-t metadata.tsv` says to use this file for the metadata.

Argument `-c 4,3` says which columns to display and sort by. This means sample and then marker. The purpose here is to group the samples logically (sorting on accession would not work), and suitable for group colouring.

Argument `-x 1` (default, so not needed) indicates the filename stem can be found in column 1, run accession.

Other files

The provided `negative_control.known.tsv` and `mock_community.known.tsv` files lists the expected species in the negative controls (none) and the mock community samples (the same 23 species). Sub-folders under `expected/` are created for each primer-pair, linking each accession name to either file as appropriate for assessing the classifier performance.

Sub-folders under `intermediate/` are used for intermediate files, a folder for each primer-pair.

4.8.2 High level overview

The high level summary is that all the samples have high coverage, much higher than most of the examples we have used. The coverage also varies between samples - making a fractional minimum abundance threshold attractive here. There is minimal off target signal (from the other primer sets), and the no template blanks have lower yields. The read counts in the blanks are high, but happily do not appear to contain nematode sequence.

Per-marker yield

We'll start by looking at the number of read-pairs found for each marker. After calling `./run.sh` you should be able to inspect these report files at the command line or in Excel.

```
$ cut -f 1,2,5-7,9,12 summary/NF1-18Sr2b.samples.onebp.tsv
<SEE TABLE BELOW>
```

Or open the Excel version `summary/NF1-18Sr2b.samples.onebp.xlsx`, and focus on those early columns:

#marker	sample	Raw FASTQ	Flash	Cutadapt	Threshold	Accepted
D3Af-D3Br	Blank	1193593	1039205	0	25	0
D3Af-D3Br	MC1	3897994	3317661	0	25	0
D3Af-D3Br	MC2	4228233	3685150	0	25	0
D3Af-D3Br	MC3	4309817	3864130	0	25	0
JB3-JB5GED	Blank	69641	62060	0	25	0
JB3-JB5GED	MC1	1236201	1157824	0	25	0
JB3-JB5GED	MC2	2160885	2058441	1	25	0
JB3-JB5GED	MC3	1204900	1139777	0	25	0
NF1-18Sr2b	Blank	260778	218813	187776	25	140063
NF1-18Sr2b	MC1	2483453	2126062	2109488	25	1394883
NF1-18Sr2b	MC2	2349364	1985981	1972923	25	1359884
NF1-18Sr2b	MC3	2435278	2088185	2070379	25	1409844
SSUF04-SSUR22	Blank	57199	46879	0	25	0
SSUF04-SSUR22	MC1	3162379	2633321	77	25	0
SSUF04-SSUR22	MC2	2790363	2370732	280	25	0
SSUF04-SSUR22	MC3	1953138	1640045	52	25	0

You should find the raw FASTQ numbers match the author's Table 5, although that omits the blanks - which happily are all much lower.

The "Flash" column reports how many of those raw FASTQ read pairs could be overlap merged into a single sequence - and our numbers range from 82% to 95% (it is easy to add this calculation in Excel). This is very different from the author's results in Table 6, although we agree that the best yield was with the JB3-JB5GED markers. Exploring the flash settings here, using `-O` or `--allow-outies` was important here to maximize yield, but that alone does not explain this discrepancy.

The "Cutadapt" column reports how many of those merged reads could be primer trimmed with the NF1-18Sr2b primers, and happily we get high numbers only for the NF1-18Sr2b samples, but low levels from the other samples. That could be barcode leakage in the demultiplexing, or actual unwanted DNA in the samples.

Then we have our "Threshold" and the final column highlighted here is the "Read count" after applying our minimum abundance threshold - and now we only get reads from the NF1-18Sr2b samples. These are all 25 specified at the command line with `-a 25` in the script, and `-f 0` to disable the fractional abundance threshold. This was done to reduce the false negatives in the mock communities to be more in line with the original analysis.

We can repeat this for the other three primer sets, and the same pattern is observed - strong signal only for the matching samples (with the blanks giving strong but lower counts), and all non-matching samples zero after the minimum abundance threshold is applied.

Blank controls

The excellent news is even at this (much lower than default) minimum abundance threshold there are no recognisable nematode sequences in any of the blanks.

Looking at the same sample reports (or the more detailed read reports), we see that while the blank samples with no PCR template control give lots of reads, where they can be identified the organisms are not seen in the mock communities. Quoting the paper:

> *Blank samples only yielded sequences of fungi and streptophyta.*

In our case, we found lots of fungi and also the genus *Urtica* (which is a green plant under streptophyta), but also some *Blastocystis* (Stramenopiles), *Cercomonas* (Rhizaria) and *Sphaerularioidea* (Opisthokonta).

```
$ for MARKER in NF1-18Sr2b SSUF04-SSUR22 D3Af-D3Br JB3-JB5GED; do \
  grep $MARKER.Blank summary/$MARKER.samples.onebp.tsv | cut -f 1,2,4; \
  done
<SEE TABLE EXCERPT BELOW>
```

Or manually looking at the four separate files - where column 4 is a text summary of the classifier output:

NF1-18Sr2b	Blank	Fungi (unknown species), Urtica sp., Unknown
SSUF04-SSUR22	Blank	Blastocystis sp., Fungi (unknown species), Unknown
D3Af-D3Br	Blank	Cercomonas sp., Fungi (unknown species), Sphaerularioidea gen. sp. EM-2016, Unknown
JB3-JB5GED	Blank	Unknown

It should stressed that all the blank samples have unknown sequences (indeed the JB3-JB5GED blank sequences are *all* reported as unknown).

4.8.3 Presence and absence

As discussed in the paper, the recognised species recovered from the mock community varied dramatically by marker. This example has been setup with the same list of 23 species expected for all the markers.

Note that three of the four reference sets lack a known sequence for *Laimaphelenchus penardi*, and most are missing more than just that species.

The `run.sh` script runs a classifier assessment over all the samples which is meaningful for the pooled results. There is then a loop to assess each marker individually on the four relevant samples only.

We can compare these results to Ahmed *et al.* (2019) Table 9.

NF1-18Sr2b

This marker has the best database coverage.

```
$ cut -f 1-5,9,11 summary/NF1-18Sr2b.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	52	60	17	191	0.57	0.597
Acrobeles sp.	0	0	3	1	0.00	1.000
Acrobeloides sp.	2	0	1	1	0.80	0.333
Alaimus sp.	1	0	2	1	0.50	0.667
Anaplectus sp.	0	0	3	1	0.00	1.000
Anatonchus tridentatus	3	0	0	1	1.00	0.000
Aphelenchoides sp.	3	0	0	1	1.00	0.000
Aporcelaimellus sp.	3	0	0	1	1.00	0.000
Criconema sp.	2	0	1	1	0.80	0.333
Ditylenchus dipsaci	3	0	0	1	1.00	0.000
Ditylenchus weischeri	0	3	0	1	0.00	1.000
Globodera achilleae	0	3	0	1	0.00	1.000
Globodera artemisiae	0	3	0	1	0.00	1.000
Globodera mexicana	0	3	0	1	0.00	1.000
Globodera pallida	0	3	0	1	0.00	1.000
Globodera rostochiensis	3	0	0	1	1.00	0.000
Globodera sp.	0	3	0	1	0.00	1.000
Globodera tabacum	0	3	0	1	0.00	1.000
Hemicycliophora sp.	1	0	2	1	0.50	0.667
Laimaphelenchus penardi	3	0	0	1	1.00	0.000
Longidorus caespiticola	3	0	0	1	1.00	0.000
Meloidogyne cf. hapla 8 JH-2014	0	3	0	1	0.00	1.000
Meloidogyne ethiopica	0	3	0	1	0.00	1.000
Meloidogyne hapla	3	0	0	1	1.00	0.000
Meloidogyne incognita	0	3	0	1	0.00	1.000
Plectus sp.	3	0	0	1	1.00	0.000
Prionchulus cf. punctatus TSH-2005	0	2	0	2	0.00	1.000
Prionchulus muscorum	0	2	0	2	0.00	1.000
Prionchulus punctatus	2	0	1	1	0.80	0.333
Pristionchus sp.	3	0	0	1	1.00	0.000
Rhabditis sp.	3	0	0	1	1.00	0.000
Steinernema carpocapsae	3	0	0	1	1.00	0.000
Steinernema monticolum	0	3	0	1	0.00	1.000
Steinernema sp.	0	3	0	1	0.00	1.000
Steinernema websteri	0	3	0	1	0.00	1.000
Trichodorus primitivus	3	0	0	1	1.00	0.000
Tripyla daviesae	0	3	0	1	0.00	1.000
Tripyla glomerans	0	0	3	1	0.00	1.000
Tripyla sp.	0	3	0	1	0.00	1.000
Tylenchus sp.	3	0	0	1	1.00	0.000
Urtica sp.	0	1	0	3	0.00	1.000
Xiphinema bakeri	0	2	0	2	0.00	1.000

continues on next page

Table 2 – continued from previous page

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
Xiphinema coxi europaeum	0	2	0	2	0.00	1.000
Xiphinema diversicaudatum	2	0	1	1	0.80	0.333
Xiphinema japonicum	0	2	0	2	0.00	1.000
Xiphinema pseudocoxi	0	2	0	2	0.00	1.000
Xiphinema vuittenezi	0	2	0	2	0.00	1.000
OTHER 34 SPECIES IN DB	0	0	0	136	0.00	0.000

We have explainable false positives as within genus conflicts in *Ditylenchus*, *Globodera*, *Meloidogyne*, *Steinernema*, *Prionchulus*, *Tripyla*, and *Xiphinema*. Note expected species *Tripyla glomerans* is not reported.

Additionally there is an unexplained FP from plant *Urtica* sp. in the blank sample.

We also have false negatives, including reporting *Anatonchus* sp. rather than *Anatonchus tridentatus*, no *Acrobeles* sp. in any of the three samples, and a few more not appearing in all the samples.

This is not performing as well as the authors' analysis:

> The NF1-18Sr2b had the highest coverage, producing 100% recovery of the sampled taxa (Table 9). All 23 taxa were detected in all three replicates, apart from *Acrobeles* and *Criconema*. They both failed to appear in one of the replicates.

Perhaps our abundance threshold is still too high?

SSUF04-SSUR22

The assess command here warns the DB lacks 10 of the expected species in the mock community, which are therefore false negatives.

```
$ cut -f 1-5,9,11 summary/SSUF04-SSUR22.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	32	6	37	37	0.60	0.573
Acrobeles sp.	0	0	3	1	0.00	1.000
Acrobeloides sp.	2	0	1	1	0.80	0.333
Alaimus sp.	3	0	0	1	1.00	0.000
Anaplectus sp.	3	0	0	1	1.00	0.000
Anatonchus tridentatus	3	0	0	1	1.00	0.000
Aphelenchoides sp.	0	0	3	1	0.00	1.000
Aporcelaimellus sp.	3	0	0	1	1.00	0.000
Blastocystis sp.	0	1	0	3	0.00	1.000
Criconema sp.	0	0	3	1	0.00	1.000
Ditylenchus dipsaci	0	0	3	1	0.00	1.000
Globodera rostochiensis	0	0	3	1	0.00	1.000
Hemicycliophora sp.	0	0	3	1	0.00	1.000
Laimaphelenchus penardi	0	0	3	1	0.00	1.000
Longidorus caespiticola	3	0	0	1	1.00	0.000
Meloidogyne hapla	0	0	3	1	0.00	1.000
Plectus sp.	3	0	0	1	1.00	0.000
Prionchulus muscorum	0	3	0	1	0.00	1.000
Prionchulus punctatus	3	0	0	1	1.00	0.000
Prionchulus sp.	0	2	0	2	0.00	1.000
Pristionchus sp.	0	0	3	1	0.00	1.000
Rhabditis sp.	0	0	3	1	0.00	1.000
Steinernema carpocapsae	3	0	0	1	1.00	0.000
Trichodorus primitivus	3	0	0	1	1.00	0.000
Tripyla glomerans	0	0	3	1	0.00	1.000
Tylenchus sp.	0	0	3	1	0.00	1.000
Xiphinema diversicaudatum	3	0	0	1	1.00	0.000
OTHER 2 SPECIES IN DB	0	0	0	8	0.00	0.000

There are false positives within the genus *Prionchulus* (wrong species), and also from *Blastocystis* sp. in the blank.

We have TP for 11 species only. The original analysis reported recovering 15 out of 23 species with this marker (Table 9), and wrote:

> In the case of the SSUF04-SSUR22 marker, eight taxa were missing from all three assignment methods. The taxa that were recovered occurred in all three replicates. With all three methods of taxonomy assignment combined, the number of correctly assigned OTUs improved to 56.

Many of our false negatives are likely due to the database coverage, with the Table 9 noting the majority of their reference sequences from NCBI RefSeq were partial - our pipeline requires full length reference amplicons.

D3Af-D3Br

The assess command here warns the DB lacks three of the expected species in the mock community, *Criconema* sp., *Laimaphelenchus penardi*, and *Steinernema carpocapsae* - which are therefore false negatives.

```
$ cut -f 1-5,9,11 summary/D3Af-D3Br.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	42	17	27	98	0.66	0.512
Acrobeles sp.	2	0	1	1	0.80	0.333
Acrobeloides sp.	0	0	3	1	0.00	1.000
Alaimus sp.	0	0	3	1	0.00	1.000
Anaplectus sp.	0	0	3	1	0.00	1.000
Anatonchus tridentatus	3	0	0	1	1.00	0.000
Aphelenchoides sp.	0	0	3	1	0.00	1.000
Aporcelaimellus sp.	3	0	0	1	1.00	0.000
Cercomonas sp.	0	1	0	3	0.00	1.000
Criconema sp.	0	0	3	1	0.00	1.000
Ditylenchus dipsaci	3	0	0	1	1.00	0.000
Globodera pallida	0	3	0	1	0.00	1.000
Globodera rostochiensis	3	0	0	1	1.00	0.000
Globodera sp.	0	3	0	1	0.00	1.000
Hemicycliophora sp.	1	0	2	1	0.50	0.667
Laimaphelenchus deconincki	0	3	0	1	0.00	1.000
Laimaphelenchus penardi	0	0	3	1	0.00	1.000
Longidorus caespiticola	3	0	0	1	1.00	0.000
Meloidogyne hapla	3	0	0	1	1.00	0.000
Plectus sp.	3	0	0	1	1.00	0.000
Prionchulus punctatus	3	0	0	1	1.00	0.000
Pristionchus sp.	3	0	0	1	1.00	0.000
Rhabditis sp.	3	0	0	1	1.00	0.000
Sphaerularioidea gen. sp. EM-2016	0	1	0	3	0.00	1.000
Steinernema carpocapsae	0	0	3	1	0.00	1.000
Trichodorus primitivus	3	0	0	1	1.00	0.000
Tripyla glomerans	3	0	0	1	1.00	0.000
Tylenchus sp.	0	0	3	1	0.00	1.000
Xiphinema bakeri	0	2	0	2	0.00	1.000
Xiphinema diversicaudatum	3	0	0	1	1.00	0.000
Xiphinema japonicum	0	2	0	2	0.00	1.000
Xiphinema sp.	0	2	0	2	0.00	1.000
OTHER 15 SPECIES IN DB	0	0	0	60	0.00	0.000

Most of the false positives are within the genus *Globodera* or *Xiphinema*, but additionally *Cercomonas* sp. and *Sphaerularioidea* gen. sp. EM-2016. Note *Laimaphelenchus deconincki* is reported instead of the expected *Laimaphelenchus penardi* here.

We have 15 species correctly identified (11 from all three samples), which exceeds authors' analysis with UTAX but falls short of their consensus:

> The 28S rDNA-based D3Af-D3Br marker assigned 70 OTUs to nematodes and recovered all taxa except *Criconema* in the consensus taxonomy. Amongst the recovered taxa, *Hemicycliophora* occurred in one of the replicates, *Acrobeles* in two, while the rest were found in all three replicates.

Note that as per the paper Table 1, accessions MG994941 and MG994928 were used for *Anatonchus tridentatus* and *Tripyla glomerans*, but required 34 and 35bp 3' extensions respectively to cover the D3Af-D3Br amplicon (missing sequenced inferred from the observed reads, and matches other nematode sequences).

JB3-JB5GED

The assess command here warns the DB lacks 20 of the expected species in the mock community, which puts the results into perspective:

```
$ cut -f 1-5,9,11 summary/JB3-JB5GED.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	9	3	60	24	0.22	0.875
Acrobeles sp.	0	0	3	1	0.00	1.000
Acrobeloides sp.	0	0	3	1	0.00	1.000
Alaimus sp.	0	0	3	1	0.00	1.000
Anaplectus sp.	0	0	3	1	0.00	1.000
Anatonchus tridentatus	0	0	3	1	0.00	1.000
Aphelenchoides sp.	0	0	3	1	0.00	1.000
Aporcelaimellus sp.	0	0	3	1	0.00	1.000
Criconema sp.	0	0	3	1	0.00	1.000
Ditylenchus dipsaci	0	0	3	1	0.00	1.000
Globodera rostochiensis	3	0	0	1	1.00	0.000
Hemicycliophora sp.	0	0	3	1	0.00	1.000
Laimaphelenchus penardi	0	0	3	1	0.00	1.000
Longidorus caespiticola	0	0	3	1	0.00	1.000
Meloidogyne hapla	3	0	0	1	1.00	0.000
Plectus sp.	0	0	3	1	0.00	1.000
Prionchulus punctatus	0	0	3	1	0.00	1.000
Pristionchus sp.	0	0	3	1	0.00	1.000
Rhabditis sp.	0	0	3	1	0.00	1.000
Steinernema abbasi	0	3	0	1	0.00	1.000
Steinernema carpocapsae	3	0	0	1	1.00	0.000
Trichodorus primitivus	0	0	3	1	0.00	1.000
Tripyla glomerans	0	0	3	1	0.00	1.000
Tylenchus sp.	0	0	3	1	0.00	1.000
Xiphinema diversicaudatum	0	0	3	1	0.00	1.000

This has performed perfectly on *Meloidogyne hapla*, *Globodera rostochiensis*, and *Steinernema carpocapsae* - although we also get false positive matches to sister species *Steinernema abbasi*.

This is better than the authors analysis, which did not find *Globodera*:

> For the COI-based JB3-JB5GED marker, even the consensus taxonomy drawn from all three assignment methods could only recover two taxa, namely Meloidogyne and Steinernema.

Pooled

The pipeline is setup to assess the pooled results expecting all 23 species in each mock community, regardless of which marker was being sequenced. i.e. This is handicapped by adding up to 9 false negatives per species.

```
$ cut -f 1-5,9,11 summary/pooled.assess.onebp.tsv
<SEE TABLE BELOW>
```

Or open this in Excel. You should find:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	135	86	141	1142	0.54	0.627
Acrobeles sp.	2	0	10	4	0.29	0.833
Acrobeloides sp.	4	0	8	4	0.50	0.667
Alaimus sp.	4	0	8	4	0.50	0.667
Anaplectus sp.	3	0	9	4	0.40	0.750
Anatonchus tridentatus	9	0	3	4	0.86	0.250
Aphelenchoides sp.	3	0	9	4	0.40	0.750
Aporcelaimellus sp.	9	0	3	4	0.86	0.250
Blastocystis sp.	0	1	0	15	0.00	1.000
Cercomonas sp.	0	1	0	15	0.00	1.000
Criconema sp.	2	0	10	4	0.29	0.833
Ditylenchus dipsaci	6	0	6	4	0.67	0.500
Ditylenchus weischeri	0	3	0	13	0.00	1.000
Globodera achilleae	0	3	0	13	0.00	1.000
Globodera artemisiae	0	3	0	13	0.00	1.000
Globodera mexicana	0	3	0	13	0.00	1.000
Globodera pallida	0	6	0	10	0.00	1.000
Globodera rostochiensis	9	0	3	4	0.86	0.250
Globodera sp.	0	6	0	10	0.00	1.000
Globodera tabacum	0	3	0	13	0.00	1.000
Hemicycliophora sp.	2	0	10	4	0.29	0.833
Laimaphelenchus deconincki	0	3	0	13	0.00	1.000
Laimaphelenchus penardi	3	0	9	4	0.40	0.750
Longidorus caespiticola	9	0	3	4	0.86	0.250
Meloidogyne cf. hapla 8 JH-2014	0	3	0	13	0.00	1.000
Meloidogyne ethiopica	0	3	0	13	0.00	1.000
Meloidogyne hapla	9	0	3	4	0.86	0.250
Meloidogyne incognita	0	3	0	13	0.00	1.000
Plectus sp.	9	0	3	4	0.86	0.250
Prionchulus cf. punctatus TSH-2005	0	2	0	14	0.00	1.000
Prionchulus muscorum	0	5	0	11	0.00	1.000
Prionchulus punctatus	8	0	4	4	0.80	0.333
Prionchulus sp.	0	2	0	14	0.00	1.000
Pristionchus sp.	6	0	6	4	0.67	0.500
Rhabditis sp.	6	0	6	4	0.67	0.500
Sphaerularioidea gen. sp. EM-2016	0	1	0	15	0.00	1.000
Steinernema abbasi	0	3	0	13	0.00	1.000
Steinernema carpocapsae	9	0	3	4	0.86	0.250
Steinernema monticolum	0	3	0	13	0.00	1.000
Steinernema sp.	0	3	0	13	0.00	1.000
Steinernema websteri	0	3	0	13	0.00	1.000

continues on next page

Table 4 – continued from previous page

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
Trichodorus primitivus	9	0	3	4	0.86	0.250
Tripyla daviesae	0	3	0	13	0.00	1.000
Tripyla glomerans	3	0	9	4	0.40	0.750
Tripyla sp.	0	3	0	13	0.00	1.000
Tylenchus sp.	3	0	9	4	0.40	0.750
Urtica sp.	0	1	0	15	0.00	1.000
Xiphinema bakeri	0	4	0	12	0.00	1.000
Xiphinema coxi europaeum	0	2	0	14	0.00	1.000
Xiphinema diversicaudatum	8	0	4	4	0.80	0.333
Xiphinema japonicum	0	4	0	12	0.00	1.000
Xiphinema pseudocoxi	0	2	0	14	0.00	1.000
Xiphinema sp.	0	2	0	14	0.00	1.000
Xiphinema vuittenezi	0	2	0	14	0.00	1.000
OTHER 41 SPECIES IN DB	0	0	0	656	0.00	0.000

As expected from the per-marker results, the false positives are largely due to species level difficulties within the genera including *Globodera*, *Steinernema*, *Tripyla*, and *Xiphinema*.

While many of the number of false negatives may be down to database coverage, it would also be worth exploring further dropping the minimum abundance threshold.

4.9 Pest Insect Mock Communities

Here we consider mock communities of up to six insect species, sequenced with three pooled markers (18S, 12S, COI), with no-template PCR control blanks:

> Batovska *et al.* (2021) Developing a non-destructive metabarcoding protocol for detection of pest insects in bulk trap catches https://doi.org/10.1038/s41598-021-85855-6 https://www.ebi.ac.uk/ena/data/view/PRJNA716058 https://zenodo.org/record/3557020 https://github.com/alexpiper/HemipteraMetabarcodingMS

This example requires creating a database covering the three different marker primers and known sequences (all of which ought to be properly curated).

4.9.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/pest_insects/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed.

FASTQ data

File `PRJNA716058.tsv` was download from the ENA and includes the FASTQ checksums, URLs, and sample metadata.

Script `setup.sh` will download the raw FASTQ files for Batovska *et al.* (2021) from https://www.ebi.ac.uk/ena/data/view/PRJNA716058

It will download 60 raw FASTQ files (30 pairs), taking 7.9 GB on disk.

If you have the `md5sum` tool installed (standard on Linux), verify the FASTQ files downloaded correctly:

```
$ cd raw_data/
$ md5sum -c MD5SUM.txt
...
$ cd ../
```

There is no need to decompress the files.

Amplicon primers & reference sequences

Three separate markers used here, as shown in the paper's Supplementary Table S2, together with the shared Illumina adaptors used.

The authors provide their reference species level sequences as a compressed FASTA file `merged_arthropoda_rdp_species.fa.gz` on the GitHub repository for the paper: https://github.com/alexpiper/HemipteraMetabarcodingMS

The worked example applies the three primer-pairs to this FASTA file to make an amplicon specific FASTA file for each marker.

Metadata

File `metadata.tsv` is based on the ENA metadata and the paper text. It has four columns:

1. run_accession, assigned by the public archive, e.g. "SRR14022295"
2. sample_alias, e.g. "100-Pool-1" or "Trap-1"
3. source, e.g. one of the mock communities like "Pool 1", or "Trap"
4. individuals, e.g. "0100" (with leading zero for sorting) or "-" for traps.

When calling THAPBI PICT, the meta data commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -x 1 -c 3,4,2
```

Argument `-t metadata.tsv` says to use this file for the metadata.

Argument `-c 3,4,2` says which columns to display and sort by. This means by source (i.e. which mock community, or environmental traps), then number of individuals in the mock, and finally the human readable sample alias. The purpose here is to group the samples logically (sorting on sample_alias would not work), and suitable for group colouring.

Argument `-x 1` (default, so not needed) indicates the filename stem can be found in column 1, run accession.

Other files

Files `mock_community_1.known.tsv`, ..., `mock_community_5.known.tsv` list the expected species in the five different mock community pools. The setup script will create symlinks using the sample names under sub-folder `expected/` pointing at the relevant community known file. This is for automatically assessing the classifier performance.

Sub-folders under `intermediate/` are used for intermediate files, a folder for each primer-pair.

4.9.2 High level overview

The high level summary is that all the samples have high coverage, much higher than most of the examples we have used. Some of the samples yield over a million reads for the COI and 12S amplicons, which with the default fractional minimum abundance threshold of 0.1% (`-f 0.001`) would mean using over 1000 reads as the threshold. This was too stringent, so the worked example reduces this to 0.01% (with `-f 0.0001`) matching the author's analysis, and dropped the default absolute abundance threshold of 100 to 50 (with `-a 50`).

Note that the rarest members of the mock communities are expected from 1 in 500 individuals (0.02%) or 1 in 1000 individuals (0.01%), which is ten times higher than the fractional abundance threshold.

Sequence yield

We'll start by looking at the number of read-pairs found for each marker. After calling `./run.sh` you should be able to inspect these report files at the command line or in Excel.

```
$ cut -f 3,6-8,10,12-14 summary/COI.samples.onebp.tsv
<SEE TABLE BELOW>
```

Or open the Excel version `summary/COI.samples.onebp.xlsx`, and focus on those early columns:

sample_alias	Raw FASTQ	Flash	Cutadapt	Threshold	Singletons	Accepted	Unique
100-Pool-1	478705	474621	109233	50	11074	86402	178
250-Pool-1	1845819	1829913	157310	50	23119	118383	251
500-Pool-1	647776	643030	51092	50	6446	36718	127
1000-Pool-1	855997	848914	66002	50	7967	49058	149
100-Pool-2	737998	732014	432826	50	29168	368249	418
250-Pool-2	2037475	2022814	1250718	126	85718	1042562	482
500-Pool-2	1908370	1895715	1231908	124	59702	1042441	442
1000-Pool-2	1068715	1060596	584017	59	33060	498955	445
100-Pool-3	950692	940342	249422	50	24964	189156	371
250-Pool-3	1631700	1615113	274422	50	39974	192944	562
500-Pool-3	923807	916621	358429	50	32221	284819	567
1000-Pool-3	1773647	1758637	468361	50	42263	374487	733
100-Pool-4	634017	628523	117499	50	14596	74799	175
250-Pool-4	2501145	2480381	441558	50	61904	324512	707
500-Pool-4	572779	568565	144488	50	18279	96537	306
1000-Pool-4	1198812	1189853	294607	50	30130	220678	470
100-Pool-5	1817929	1800594	434739	50	45224	329015	660
250-Pool-5	1632786	1617219	440995	50	58159	328842	729
500-Pool-5	807060	801471	321428	50	30944	247519	484
1000-Pool-5	1423279	1411512	332286	50	32751	255309	584
Trap-1	1759819	1719671	110882	50	19740	73024	251

continues on next page

Table 5 – continued from previous page

sample_alias	Raw FASTQ	Flash	Cutadapt	Threshold	Singletons	Accepted	Unique
Trap-10	2445993	2420303	308371	50	58670	204842	480
Trap-2	1127739	1107970	110856	50	24385	55757	92
Trap-3	2422054	2366037	161686	50	30631	110043	268
Trap-4	742893	732907	63107	50	11933	35225	77
Trap-5	3437292	3346620	346696	50	71464	208989	542
Trap-6	697389	689125	91284	50	17153	57037	149
Trap-7	2853448	2820200	223330	50	31011	169121	319
Trap-8	2196646	2161966	146646	50	28814	92632	220
Trap-9	2065455	2049024	70591	50	14636	40131	109

The marker specific tables show the threshold applied was usually 50, the default absolute value set via `-a 50` at the command line. Occasionally this has been increased to 0.1% of the sequences matching the primers for this marker, set via `-f 0.0001` at the command line.

The numbers are similar for the 12S and 18S markers, or pooling them all:

```
$ cut -f 3,6,7,13,14 summary/pooled.samples.onebp.tsv
<SEE TABLE BELOW>
```

Again, alternatively open Excel file `summary/pooled.samples.onebp.xlsx`, and focus on those early columns:

sample_alias	Raw FASTQ	Flash	Accepted	Unique
100-Pool-1	478705	474621	371045	703
250-Pool-1	1845819	1829913	1508292	689
500-Pool-1	647776	643030	522396	800
1000-Pool-1	855997	848914	692639	950
100-Pool-2	737998	732014	587902	886
250-Pool-2	2037475	2022814	1243165	837
500-Pool-2	1908370	1895715	1551757	1142
1000-Pool-2	1068715	1060596	863574	1024
100-Pool-3	950692	940342	684297	1479
250-Pool-3	1631700	1615113	1158575	1241
500-Pool-3	923807	916621	697552	1457
1000-Pool-3	1773647	1758637	1366298	1993
100-Pool-4	634017	628523	451801	879
250-Pool-4	2501145	2480381	1867605	1171
500-Pool-4	572779	568565	416456	925
1000-Pool-4	1198812	1189853	918004	1660
100-Pool-5	1817929	1800594	1369274	1918
250-Pool-5	1632786	1617219	1128901	1475
500-Pool-5	807060	801471	603390	1276
1000-Pool-5	1423279	1411512	1104412	1716
Trap-1	1759819	1719671	392775	919
Trap-10	2445993	2420303	492325	1079
Trap-2	1127739	1107970	129956	273
Trap-3	2422054	2366037	427533	953
Trap-4	742893	732907	232800	403
Trap-5	3437292	3346620	486282	1177
Trap-6	697389	689125	80003	170
Trap-7	2853448	2820200	1158684	842

continues on next page

Table 6 – continued from previous page

sample_alias	Raw FASTQ	Flash	Accepted	Unique
Trap-8	2196646	2161966	683669	1024
Trap-9	2065455	2049024	1352408	689

The "Accepted" column is the number of reads matching the primer pairs and passing our abundance thresholds. The fraction accepted varies from 61% to 82% for the mock community samples, but is considerably lower for the environmental traps, varying from 11% to 65%. Much of that would be noise and trace level environmental DNA.

The "Unique" column is the number of accepted unique sequences. For the mock communities this should be up to 18 with at most six species each, and three markers. The observed counts are much higher, so we might want to denoise, or and/or raise the abundance threshold higher. Dropping it further does raise the false positive rate inferred from the mock communities.

4.9.3 Presence and absence

This example includes mock communities which are a controlled setup where we know what the classifier ought ideally to report for every sample - and all their expected marker sequences are in the classification database.

There are five different mock communities, made up with different numbers of individuals. Running an overall assessment on the pooled species assignments from all three markers we have both false positives, and false negatives:

```
$ cut -f 1-5,9,11 summary/pooled.assess.onebp.tsv
<SEE TABLE BELOW>
```

As a table:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	85	21	3	1104331	0.88	0.220
Acizzia alternata	16	1	0	3	0.97	0.059
Acizzia solanicola	16	1	0	3	0.97	0.059
Bactericera cockerelli	10	0	2	8	0.91	0.167
Diuraphis noxia	11	3	1	5	0.85	0.267
Metopolophium dirhodum	16	0	0	4	1.00	0.000
Rhopalosiphum nymphaeae	0	16	0	4	0.00	1.000
Rhopalosiphum padi	16	0	0	4	1.00	0.000
OTHER 55215 SPECIES IN DB	0	0	0	1104300	0.00	0.000

Our 3 false negatives on the pooled results are *Bactericera cockerelli* (2 cases, `1000-Pool-1` and ``250-Pool-4` matching the authors, with `500-Pool-4` just passing at 62 reads), and *Diuraphis noxia* (1 case, `500-Pool-3`). The authors also reported `500-Pool-3` missing *D. noxia*, here it just passes the threshold at 54 reads.

Most of the false positives are 16 cases of *Rhopalosiphum nymphaeae*, which is unfortunately indistinguishable from community member *R. padi* with the 12S marker (see the `thapbi_pict conflicts ...` output).

Likewise splitting *Acizzia alternata* and *solanicola* is not possible with 18S, but we still have unwanted *Acizzia alternata* and *solanicola* from 12S in `500 Pool 2`, with 53 and 96 reads respectively. These counts are low enough to consider raising the abundance threshold(s) to exclude them. In both cases they match the dominant 12S sequences from the other controls and could be due to Illumina tag switching?

The remaining false positives are 3 cases of *Diuraphis noxia* via the 18S marker, and in one case via the COI marker.

The authors only reported false positives for *Diuraphis noxia* in one sample, `1000-Pool-1` for both COI and 18S (Figure 3), traced to an unwanted nymph specimen (Figure 4). We see that too, but have two other false positives:

```
$ cut -f 3,37,38 summary/pooled.samples.onebp.tsv
<SEE TABLE BELOW>
```

Note 576 unwanted reads in `1000-Pool-1` for *Diuraphis noxia* (consistent with the author's analysis), but also 503 reads in `100-Pool-5` and 63 in `1000-Pool-5` with a fuzzy match for *Diuraphis noxia* and/or *Metopolophium dirhodum*:

sample_alias	Diuraphis noxia	Diuraphis noxia;Metopolophium dirhodum
100-Pool-1	0	0
250-Pool-1	0	0
500-Pool-1	0	0
1000-Pool-1	576	0
100-Pool-2	1630	0
250-Pool-2	1454	0
500-Pool-2	1660	0
1000-Pool-2	228	0
100-Pool-3	2705	126
250-Pool-3	4059	0
500-Pool-3	0	0
1000-Pool-3	94	0
100-Pool-4	6446	113
250-Pool-4	5701	0
500-Pool-4	742	0
1000-Pool-4	684	54
100-Pool-5	0	503
250-Pool-5	0	0
500-Pool-5	0	0
1000-Pool-5	0	63
Trap-1	25009	135
Trap-10	126570	53
Trap-2	106	0
Trap-3	272	0
Trap-4	1351	0
Trap-5	4235	0
Trap-6	16758	0
Trap-7	2733	0
Trap-8	99678	53
Trap-9	37358	0

Consulting the read report, these *Diuraphis noxia* false positives from `100-Pool-5` (503 copies) and `1000-Pool-5` (just 63 copies) are from the same 18S sequence:

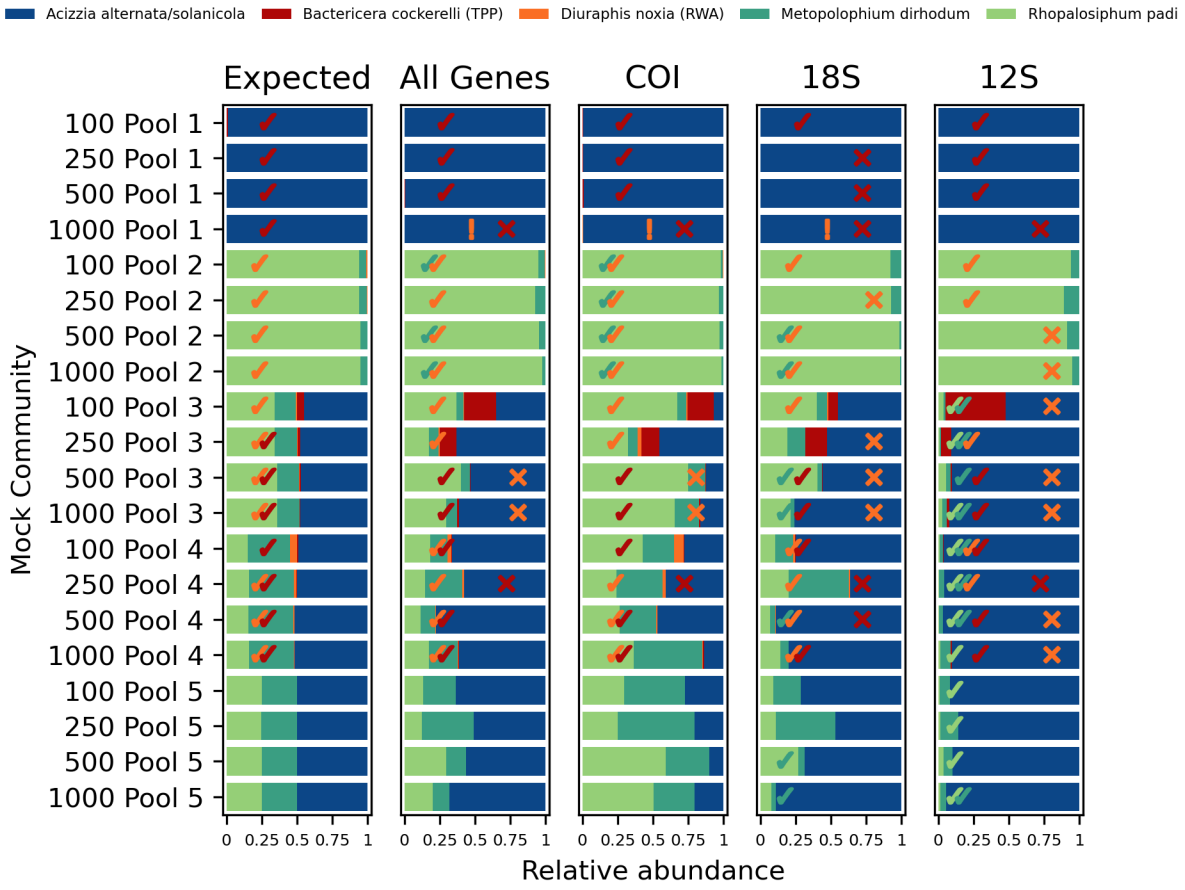
```
>d153aa679f3c184a2790cd26aac9c784
CCGCATTAAGGTGAAACCGCGAAAGGCTCATTAAATCAGTTGTGGTTCCTTAGATCGTACCCAAGTTACTTGGATAACTG
TGGTAATTCTAGAGCTAATACATGCCGACAGAGTTCCGACCGTCGCGGCGCCCTCGGGCGTCGCGCGCGGGAGGAACGCT
TTTATTAGATCAAAACCGGCCCGTCGCGGCGCGCTTCGTGCGCGTCCCGATCGCGGCCCGCGCAAAGACCTGGTGACTCT
GAATAACTTCGAGCTGATCGCACGGTCTCCGTACCGGCGACGCATCTTTCAAAT
```

With over 500 copies in `100-Pool-5` this cannot be dismissed as a difference in noise filtering versus the authors' original analysis. Querying this on NCBI BLAST confirms it to be 1bp away from multiple *Diuraphis noxia* accessions, and a *Metopolophium dirhodum* voucher sequence (as in the DB here), but also third species via a sequence labelled as *Acyrthosiphon pisum*. Rather than reporting multiple conflicting species, the author's pipeline likely assigned a lower rank?

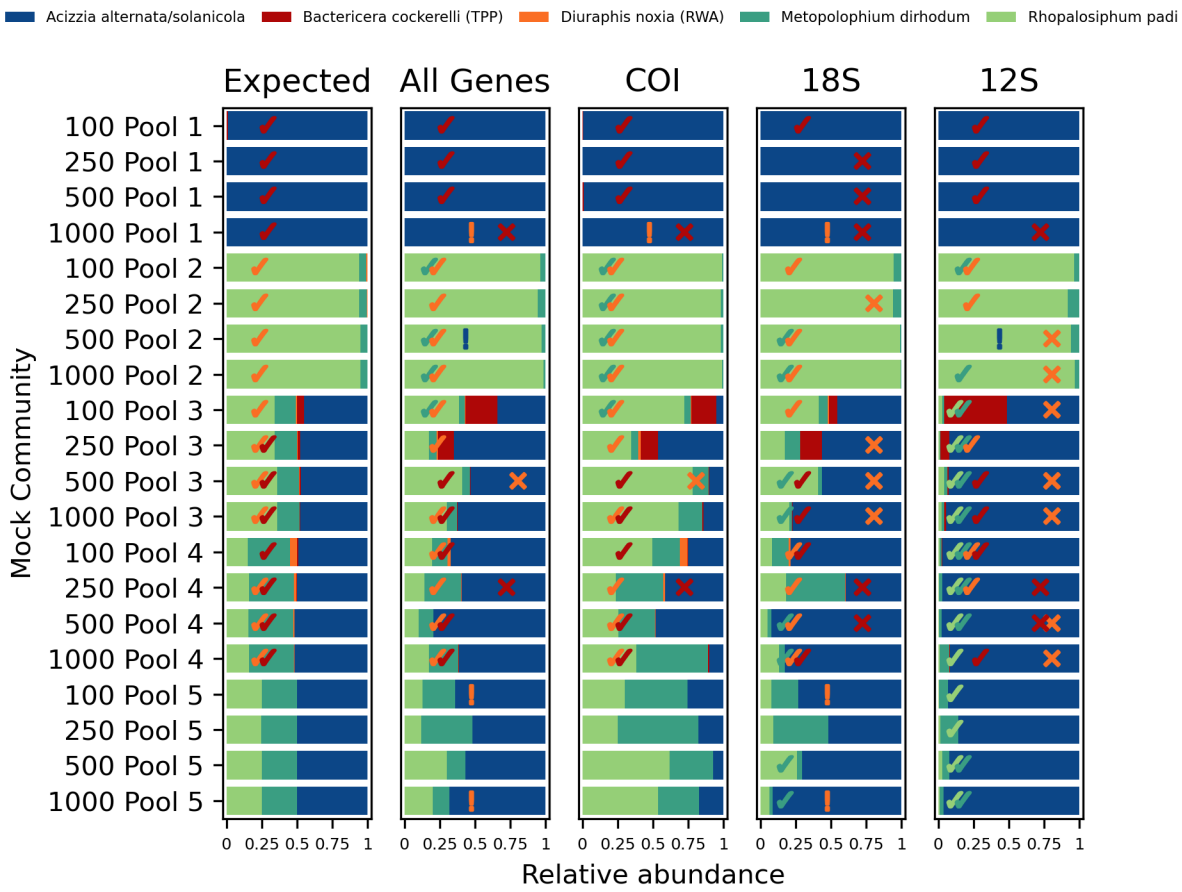
Visualisation

We can look at this visually by reproducing Figure 3 from the original paper. The authors provided their R based analysis, from which I have exported the numbers used to draw the figure (see `figure3original.R`) giving a simple tab-separated file (`figure3original.tsv`). Likewise the Python script `figure3reproduction.py` will produce an equivalent table using the output from THABPI PICT (`figure3reproduction.tsv`). Finally, Python script `recreate_figure3.py` uses MatPlotLib to reproduce an annotated recreation of the original.

Original analysis:



This re-analysis:



In the original paper the false positives and false negatives were marked with pluses and minus in colour coded circles, and these were added by hand. Here this annotation is automated, but is less aesthetically pleasing. The false negatives get a cross, false positives are shown with an exclamation mark, and furthermore low abundance (under 5%) true positives get a tick. Again, these are all species coloured.

Overall this seems to show very good agreement with the published analysis.

4.10 Endangered Species Mixes 16S etc

This is the most complicated of the examples considered, where most of the samples are "Experimental mixtures" of multiple plants and animals (plus two traditional medicine mixtures where the exact content is unknown), which have all been sequenced with about a dozen different primer pairs for multiple metabarcoding markers including 16S, COI, cyt-b, matK, rbcL, trnL and ITS2:

> Arulandhu *et al.* (2017) Development and validation of a multi-locus DNA metabarcoding method to identify endangered species in complex samples. https://doi.org/10.1093/gigascience/gix080

This example requires creating a database of multiple markers (all of which ought to be properly curated). Both per-marker reports and a pooled report are generated by the pipeline.

4.10.1 Marker data

Either clone the THAPBI PICT source code repository, or decompress the latest source code release (`.tar.gz` file). You should find it contains a directory `examples/endangered_species/` which is for this example.

Shell scripts `setup.sh` and `run.sh` should reproduce the analysis discussed - although the documentation will take you though this step by step.

Under the `intermediate/` folder will be a subdirectory for each of the primer settings, and the primer name is used as a prefix for the reports in `summary/`.

Compared to the other examples, there is an additional `tmp_merged/` subfolder which contains gzipped FASTA files after quality trimming and merging overlapping paired reads into single sequences - but prior to applying the various primers and abundance thresholds.

FASTQ data

File `PRJEB18620.tsv` was download from the ENA and includes the raw data checksums, URLs, but lacks any sample metadata.

Script `setup.sh` will download the raw FASTQ files for Arulandhu *et al.* (2017) from https://www.ebi.ac.uk/ena/data/view/PRJEB18620

It will download 354 raw FASTQ files (177 pairs), taking about 6.5GB on disk. The 177 sequenced samples are made up of 17 experimental mixtures (including only two with replicates, 1.1GB) and 160 inter-laboratory trials (16 samples repeated in 10 laboratories, 5.4GB).

This script first downloads files from the ENA under `raw_downloads/` (a mix of `*.zip` and `*.fastq.gz` files), and then sets up consistently named and compressed entries under `raw_data/*.fastq.gz` instead.

If you have the `md5sum` tool installed (standard on Linux), verify the files downloaded correctly:

```
$ cd raw_download/
$ md5sum -c MD5SUM.txt
...
$ cd ..
```

There is no need to decompress the files.

Amplicon primers & reference sequences

All the samples were all amplified with a dozen primers (see Table 1). To interpret the data properly you would need a well curated database for each marker - FASTA files are provided to build a rudimentary database.

Files `references/*.fasta` were compiled by hand on an *ad hoc* basis to use for pre-trimmed reference databases. They *should not* be used as is in any serious analysis. In many cases ambiguous matches have been omitted in preference of just species expected in the control mixtures. For example, only recording *Brassica napus* and *Brassica oleracea* despite some markers being shared by *Brassica juncea* or *Brassica nigra* etc. In more extreme cases, markers are clearly not even genus specific, but again only the control mixture representative appears - e.g. *Carica papaya*, *Glycine max*, *Gossypium hirsutum*, *Lactuca sativa*, *Solanum lycopersicum*. Deliberately reducing the false positives from these ambiguous marker sequences was done for illustrative purposes only.

Note that false positives remain, for example an ITS2 sequence most likely from *Lactuca sativa* in the control mixture is just one base pair away from a published sequence from that species (KM210323.1), but perfectly matches published sequences from *Lactuca altaica*, *L. serriola* and *L. virosa*.

Metadata

The sample metadata on the ENA is minimal, although the NCBI SRA has longer descriptions. For example run `ERS1545972` from sample `SAMEA80893168` aka `EM_1` has title "Experimental mixture 1" but only the NCBI has description "Experimental mixture containing 99% Bos taurus and 1% Lactuca sativa". Or, run `ERS1546502` from sample `SAMEA81290668` aka `S1_Lab_1` has title "Interlaboratory trial" while the NCBI also has the description "Experimental mixture containing 1% Zea mays, 1% Glycine max, 1% Aloe variegata, 1% Dendrobium sp., 1% Huso Dauricus, 1% Crocodylus niloticus, 47% Brassica oleracea and 47% Bos taurus, in dry weight percentages".

File `PRJEB18620.tsv` with the descriptions on the NCBI SRA, supplemented by Table 7, was used to write `metadata.tsv`, which has the following columns:

1. run_accessions, e.g. "ERR1824060;ERR1824061;...;ERR1824075"
2. run_names, e.g. "EM_1" or "S1_Lab_1;S1_Lab_2;...;S1_Lab_16"
3. group, "Experimental mixture" or "Interlaboratory trial"
4. sample, e.g. "EM_1" or "S1"
5. description, e.g. "Experimental mixture containing 1% Zea mays, 1% Glycine max, 1% Aloe variegata, 1% Dendrobium sp., 1% Huso Dauricus, 1% Crocodylus niloticus, 47% Brassica oleracea and 47% Bos taurus, in dry weight percentages."

Note we have a single row for each set of replicates (two cases in the initial "Experimental mixture" set, and 16 laboratories for each of the 10 "Interlaboratory trial" samples), cross referenced to the individual runs with semi-colon separated lists in columns 1 (accession) and 2 (filename).

When calling THAPBI PICT, the metadata commands are given as follows:

```
$ thapbi_pict ... -t metadata.tsv -c 3,4,5 -x 2 -g 4
```

Argument `-c 3,4,5` says which columns to display and sort by. This means group, sample, description. Given the sample prefix naming, putting the group first is not essential for sorting, but is logical.

Argument `-x 2` indicates the filename stem can be found in column 2. Unlike most of the worked examples, we are not using the accession filenames here.

Argument `-g 4` means assign colour bands using sample. This gives 15 thin bands for the "Experimental mixture" set, and then 10 wide bands for the "Interlaboratory trial" samples. By chance the two traditional medicine samples both get wide green bands in the Excel reports.

Other files

Files `expected/*.known.tsv` were compiled by hand from the species content of the experimental samples (using the PRJEB18620 sample descriptions on the NCBI and Table 7).

4.10.2 Universal animal DNA barcodes and mini-barcodes

For 16S, COI and cyt-b the paper used two targets, a long barcode and a shorter mini-barcode. The same names have been used in the `run.sh` script provided, the output of which is referred to below.

16S - long marker

The 16S primer set output is disappointing at the default abundance threshold, with only a single unique sequence observed - I suspect the long product size is part of the issue, it must be at the upper limit for overlapping MiSeq read pairs?

```
$ grep -v "^#" summary/16S.tally.tsv | cut -f 1,179
16S/1f2b15d58f9f40b862486676809d4744_20189 ␣
↪CACCTCCAGCATTCCCAGTATTGGAGGCATTGCCTGCCCAGTGACAACTGTTTAACGGCCGCGGTATCCTGACCGTGCAAAGGTAGCATAATCATTTGTTC
```

This perfectly matches *Bos taurus* and was found in most but not all of the samples expected - perhaps the default abundance threshold is too high?

Mini-16S - short marker

The output from the Mini-16S marker is far more diverse, with 84 unique sequences:

```
$ grep -c -v "^#" summary/Mini-16S.tally.tsv
84
```

The most common is again a perfect match to *Bos taurus*, which this time has no false negatives (but two false positives?).

We have all the expected *Sus scrofa* matches, and some of *Gallus gallus* and *Anguilla anguilla* expected in six samples. *Crocodylus niloticus* is also found but at far lower levels than expected.

We do see *Homo sapiens*, but happily only in the traditional medicine samples (multiple replicates within `S3` and `S8`). Within those samples, the laboratory 16 replicates `S3_Lab_16` and `S8_Lab_16` also had *Rattus tanezumi* and *Rattus norvegicus* too, respectively.

Overall, again perhaps the default abundance threshold is too high?

COI - long marker

Assuming I understood the paper correctly, this used a pool of four left primers and four right primers. That is not easily handled with THAPBI PICT at the time of writing.

Mini-COI - short marker

The output from the Mini-COI marker is quite diverse, with 22 unique sequences:

```
$ grep -c -v "^#" summary/Mini-COI.tally.tsv
22
```

The species matches are all reasonable, it detects all the *Pieris brassicae*, most of the *Bos taurus*, *Pleuronectes platessa*, *Sus scrofa*, many of the *Huso dauricus* and *Gallus gallus*.

We have unexpected *Acipenser schrenckii*, which was also found in the paper and explained due to sample preparation.

There are also plenty of unclassified sequences from the traditional medicine samples, based on an NCBI BLAST search many are likely from undescribed fungi.

cyt-b - long marker

This gave no sequences at the default abundance threshold, nor at 50. Dropping to 10 we get a modest number of hits - the only perfect match was unfortunately to plants in the Asteraceae family.

Mini-cyt-b - short marker

The output from the Mini-COI marker had only 17 unique sequences:

```
$ grep -c -v "^#" summary/Mini-cyt-b.tally.tsv
17
```

This found all the expected *Sus scrofa* and *Meleagris gallopavo*, and most *Bos taurus*, *Crocodylus niloticus*, *Huso dauricus* and some of the *Anguilla anguilla*.

As above, we have explained false matches for *Acipenser schrenckii*, and again *Homo sapiens* in the traditional medicine but also in `EM_8`.

4.10.3 Universal plant DNA barcodes and mini-barcodes

As in the animal primers, for rbcL the paper used two targets, a long barcode and a shorter mini-barcode. The same names have been used in the `run.sh` script provided, the output of which is referred to below.

matK

The paper described two sets of primers for matK, although only one was used for the MiSeq sequencing. This gave no sequences at the default abundance threshold, dropping to 50 showed three uniques sequences in three files, and even dropping to 10 only gave results from `EM_2`, `EM_14` and `S8`.

NCBI BLAST of these sequence gave no perfect matches, but suggested *Sanguisorba* sp. was present, noted in the original paper for `S8` which is one of the traditional medicine samples.

rbcL - long target

Using our default abundance threshold and the author's minimum length of 140bp, we got no sequences at all. Allowing a minimum length of 100 (our default) gave the following sequence and a one SNP variant, all from `S3`:

```
>3ec67342f519461a0ad40fef436b1b1d
GACTGCGGGGTTCAAAGCTGGTGTTAAAGATTATAGATTGACGTATTATACTCCTGAATTGGGGTTATCCGCTAAGAATT
ACGGTAGAGCAGTTTATGAATGTCTT
```

The best NCBI BLAST matches are *Astragalus*, but with a break point. The authors of the original paper report finding *Astragalus danicus* in `S3`.

Mini-rbcL - short target

This was by far and above the most diverse in terms of unique sequences recovered:

```
$ grep -c -v "^#" summary/Mini-rbcL.tally.tsv
278
```

We see expected plant species like *Lactuca sativa*, *Brassica oleracea*, *Aloe variegata* and *Dendrobium sp.* - exactly how they are classified depends critically on how the database is built.

The traditional medicine samples have multiple unknown sequences likely of plant origin.

The edit-graph is the most complicated of those in this dataset - not simply in terms of the number of nodes. This marker needs more careful review before using THAPBI PICT's default `onebp` classifier.

trnL-UAA

Not very diverse, only eight unique sequences recovered:

```
$ grep -c -v "^#" summary/trnL-UAA.tally.tsv
8
```

We see lots of *Brassica*, the difficulties with *Brassica oleracea* vs *Brassica napus* (and the genus in general) are discussed in the paper too.

trnL-P6-loop

Initially I saw no sequences with this marker, even disabling the abundance threshold. This was strange, however easily explained - quoting the paper:

> We implemented a minimum DNA barcode length of 200 nt, except for DNA barcodes with a basic length shorter than 200 nt, in which case the minimum expected DNA barcode length is set to 100 nt for ITS2, 140 nt for mini-rbcL, and 10 nt for the trnL (P6 loop) marker.

Therefore in `run.sh` we have changed the THAPBI PICT minimum length from 100 (our default) to 10 for this marker - and now get lots, over a hundred unique sequences:

```
$ grep -c -v "^#" summary/trnL-P6-loop.tally.tsv
134
```

We find this dominated by *Brassica oleracea* in most samples. However, at our default abundance threshold we do not find *Cycas revoluta* which is consistent with the original analysis reporting this at very low abundance.

Our reference set here has *Aloe reynoldsii* sequences, but none for the expected entry *Aloe variegata*.

An obvious false positive here is *Cullen* sp. which like the authors we found in the `S3` traditional medicine, but also unexpectedly in all the `S1` samples.

ITS2

Quite diverse, with over fifty unique sequences recovered:

```
$ grep -c -v "^#" summary/ITS2.tally.tsv
59
```

Finds all the *Brassica* and *Echinocactus* sp., most of the *Euphorbia* sp.

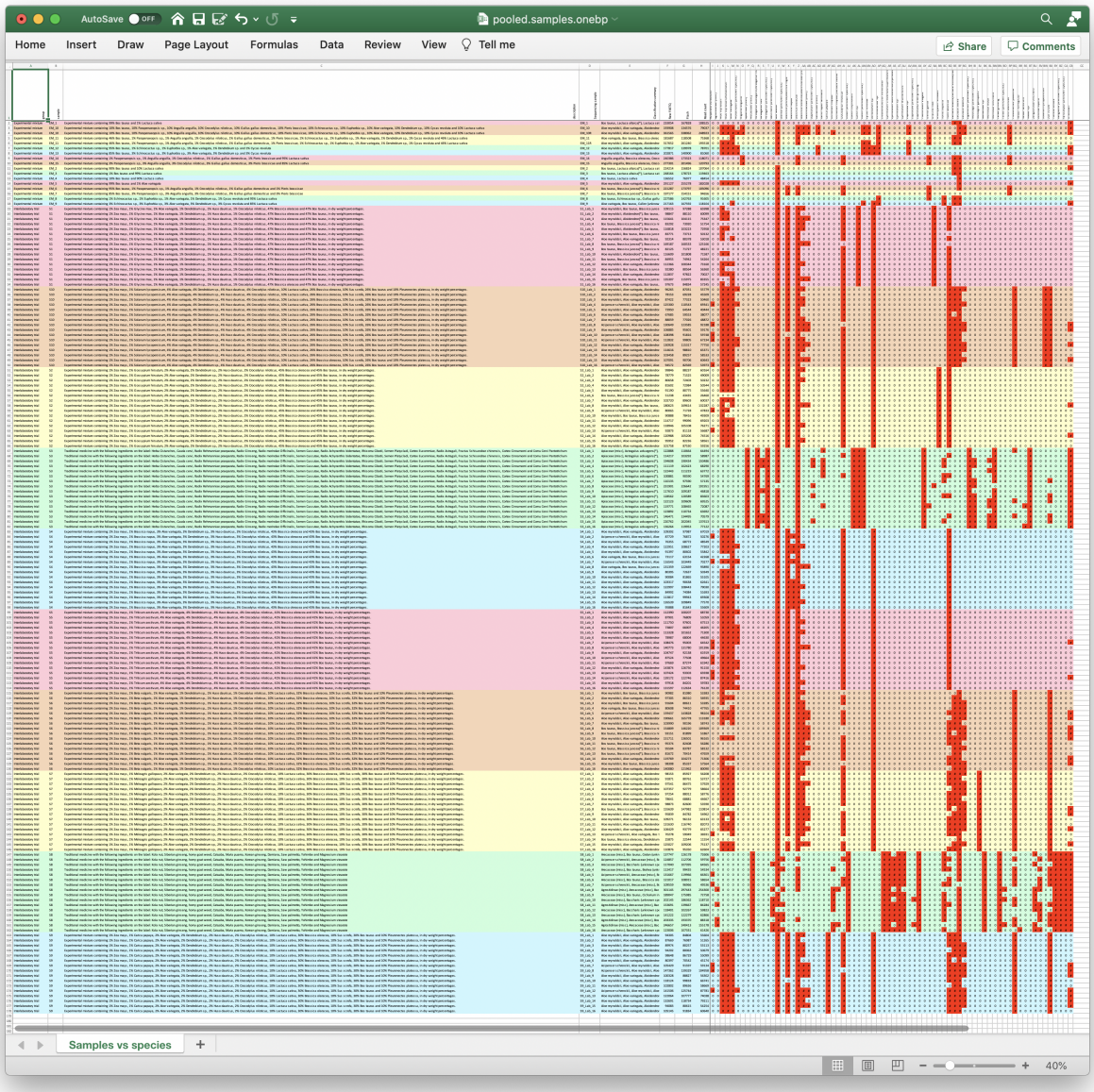
We do see unexpected matches to *Lactuca* sp. where *Lactuca sativa* was in the experimental mixture. The dominant sequence present is just one base pair away from a published sequence from that species (KM210323.1), but perfectly matches published sequences from *Lactuca altaica*, *L. serriola* and *L. virosa* - and that is what was in the sample database. If you open the associated edit-graph file (`ITS2.edit-graph.onebp.xgmml`) in Cytoscape, you can see this quite clearly.

4.10.4 Pooled animal and plant DNA barcodes

We have very briefly reviewed the output of each of the animal and plant markers, noting some have no sequences at the THAPBI PICT default minimum abundance threshold. Now we discuss the pooled results.

Sample report

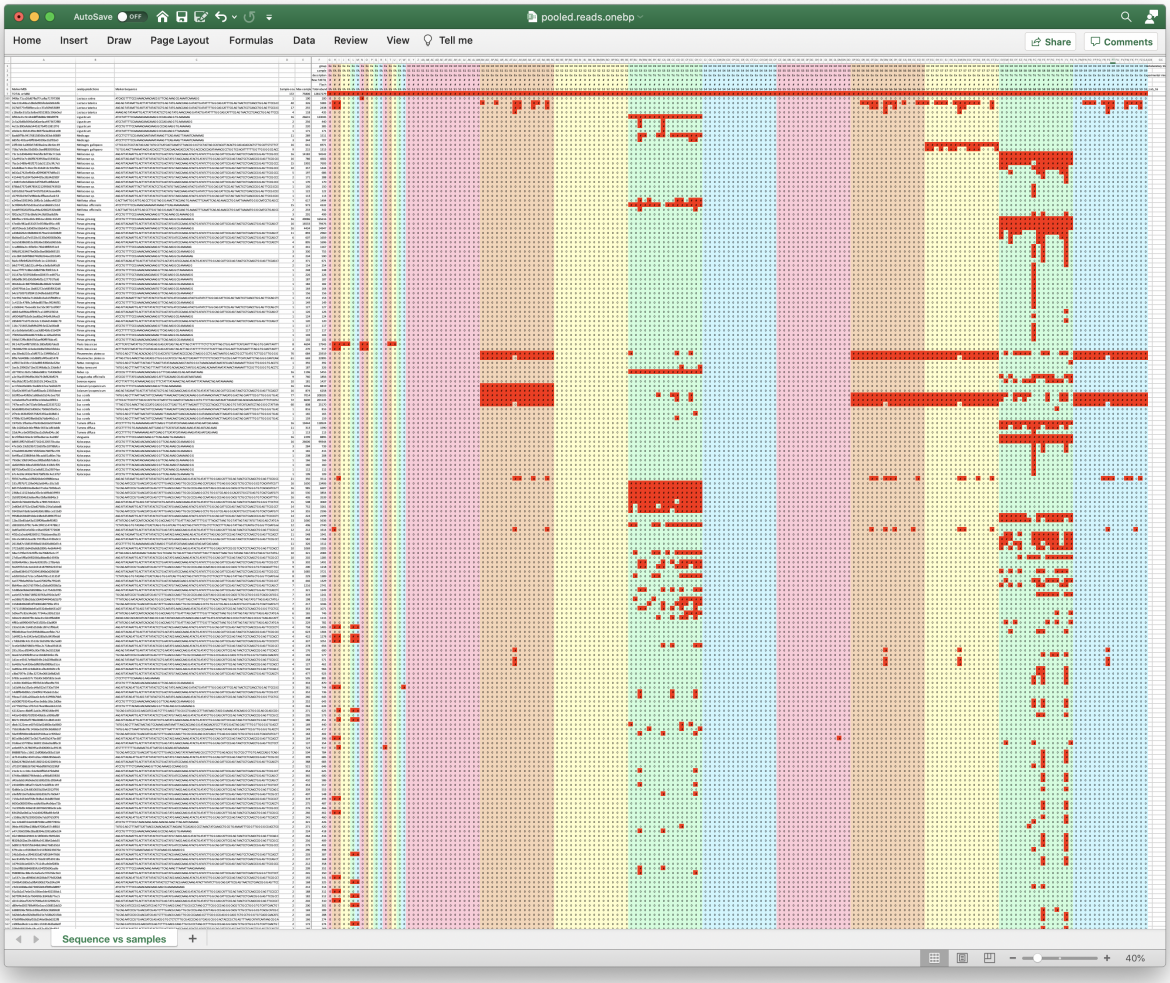
Please open the `summary/pooled.samples.onebp.xlsx` sample report, zoomed out you should have something like this:



The final column is the unknowns - and even at this zoom it is possible to see a solid red region for the two traditional medicine samples (wide green background bands).

Read report

To look at the unknown reads see `summary/pooled.reads.onebp.xlsx`. Sorting be the species prediction and zooming out should show something like this where the top half of the rows are those sequences with a species prediction. It is clear that the majority of the unknown sequences are from the two traditional medicine samples (wide green bands):



Overall the replicates are reassuringly consistent - look at neighbouring rows/columns within the colour bands in the two reports.

Pooled classifier assessment

The automated model assessment output in `summary/pooled.assess.onebp.tsv` is also worth review. Note this only looks at the experimental mixtures where there is a ground truth (S1, S2, S4, S5, S6, S7, S9 and S10) - not the traditional medicine samples where the true species content is unknown.

```
$ cut -f 1-5,9,11 summary/pooled.assess.onebp.tsv
<SEE TABLE BELOW>
```

Working at the command line or using Excel should show the following:

#Species	TP	FP	FN	TN	F1	Ad-hoc-loss
OVERALL	1058	727	240	7699	0.69	0.478
Acipenser schrenckii	0	20	0	123	0.00	1.000
Aloe reynoldsii	0	114	0	29	0.00	1.000
Aloe variegata	110	0	25	8	0.90	0.185
Anguilla anguilla	3	0	3	137	0.67	0.500
Beta vulgaris	0	0	16	127	0.00	1.000
Bos taurus	139	2	0	2	0.99	0.014
Brassica juncea	0	127	0	16	0.00	1.000
Brassica napus	7	0	9	127	0.61	0.562
Brassica nigra	0	127	0	16	0.00	1.000
Brassica oleracea	128	6	0	9	0.98	0.045
Brassicaceae (misc)	0	70	0	73	0.00	1.000
Cactaceae (misc)	0	3	0	140	0.00	1.000
Carica papaya	16	0	0	127	1.00	0.000
Crocodylus niloticus	122	0	12	9	0.95	0.090
Cullen sp.	0	16	0	127	0.00	1.000
Cycas revoluta	3	0	3	137	0.67	0.500
Dendrobium sp.	131	0	3	9	0.99	0.022
Echinocactus sp.	6	0	0	137	1.00	0.000
Euphorbia sp.	3	0	3	137	0.67	0.500
Gallus gallus	6	1	0	136	0.92	0.143
Glycine max	16	0	0	127	1.00	0.000
Gossypium hirsutum	16	0	0	127	1.00	0.000
Homo sapiens	0	2	0	141	0.00	1.000
Huso dauricus	112	0	16	15	0.93	0.125
Lactuca altaica	0	66	0	77	0.00	1.000
Lactuca sativa	74	2	0	67	0.99	0.026
Lactuca serriola	0	66	0	77	0.00	1.000
Lactuca tatarica	0	39	0	104	0.00	1.000
Lactuca virosa	0	66	0	77	0.00	1.000
Meleagris gallopavo	16	0	0	127	1.00	0.000
Parapenaeopsis sp.	0	0	6	137	0.00	1.000
Pieris brassicae	6	0	0	137	1.00	0.000
Pleuronectes platessa	64	0	0	79	1.00	0.000
Solanum lycopersicum	16	0	0	127	1.00	0.000
Sus scrofa	64	0	0	79	1.00	0.000
Triticum aestivum	0	0	16	127	0.00	1.000
Zea mays	0	0	128	15	0.00	1.000
OTHER 31 SPECIES IN DB	0	0	0	4433	0.00	0.000

Most of the false positives (FP) are alternative genus level matches in *Brassica* and *Lactuca* (as discussed in the paper). The two sequences we recorded in the Mini-rbcL reference set as the family Brassicaceae are likely also *Brassica*. The trnL-P6-loop marker had references for *Aloe reynoldsii* but these matches are most likely from *Aloe variegata*.

A couple of the unique sequences are in the Mini-rbcL reference as the family Cactaceae, and since they only appeared in the experimental mixes, these are likely *Echinocactus* sp. or *Euphorbia* sp.

There are more interesting FP for *Acipenser schrenckii* (the authors found this was accidentally included from the *Huso dauricus* caviar used), human (*Homo sapiens*, presumed laboratory contamination), and finally *Lactuca sativa*, cow (*Bos taurus*) and chicken (*Gallus gallus*) which the authors traced to cross-contamination during sample preparation or DNA isolation.

Why do *Cullen* sp. show up in `S1` from the trnL P6 loop marker (as well as `S3` which the authors found too, see their

Table 8)?

If the sample database had been more inclusive there would have been many more false positives. For example, the trnL-UAA sequence perfectly matching AP007232.1 *Lactuca sativa* is also a perfect match for MK064549.1 *Luisia teres*. Similarly, the Mini-rbcL sequence perfectly matching AP012989.1 *Brassica nigra* and MG872827.1 *Brassica juncea* also matches MN056359.2 *Raphanus sativus* (and more). This demonstrates the difficulties in curating an appropriate marker database - and the content should depend in part on your target samples.

Currently the provided references sequences (and thus classification databases used) lack any markers for *Beta vulgaris*, *Parapenaeopsis* sp., *Triticum aestivum* or *Zea mays*. Most of these were present at only a few percent dry weight, and are likely present below the default minimum abundance threshold. This explains the false negatives.

Conclusion

It appears that the THAPBI PICT default minimum abundance threshold of 100 reads is too stringent for detecting all the markers in a complex pool like this. Including negative sequencing controls would help set an objective lower bound.

There also appear to be marker sequences in these control samples which have not yet been published, which would help by filling in gaps in the reference set used for classification.

Also note we did not look at the multi-primer COI long marker, and perhaps the default `onebp` classifier is not appropriate for the Mini-rbcL marker.

- *Environmental Phytophthora ITS1* - A simple example using the default primers and database. Based on a paper from earlier in the THAPBI Phyto-Threats project:

  > Riddell *et al.* (2019) Metabarcoding reveals a high diversity of woody host-associated *Phytophthora* spp. in soils at public gardens and amenity woodlands in Britain. https://doi.org/10.7717/peerj.6931

- *Environmental Oomycetes ITS1* - An example where the defaults can be used, but ideally requires a different primer pair and a custom database. Based on:

  > Redekar *et al.* (2019) Diversity of *Phytophthora*, *Pythium*, and *Phytopythium* species in recycled irrigation water in a container nursery. https://doi.org/10.1094/PBIOMES-10-18-0043-R

- *Drained fish ponds 12S* - An example with a single marker and custom database. Based on:

  > Muri *et al.* (2020) Read counts from environmental DNA (eDNA) metabarcoding reflect fish abundance and biomass in drained ponds. https://doi.org/10.3897/mbmg.4.56959

- *Fungal Mock Community ITS1 & 2* - An example with multiple markers (including two sequenced together) requiring separate primers settings and databases, based on:

  > Bakker (2018) A fungal mock community control for amplicon sequencing experiments. https://doi.org/10.1111/1755-0998.12760

- *Great Lakes Mock Community 16S* - An example with two mitochondrial markers (sequenced separately), with mock communities, where we focus on the minimum abundance threshold. Based on:

  > Klymus *et al.* (2017) Environmental DNA (eDNA) metabarcoding assays to detect invasive invertebrate species in the Great Lakes. https://doi.org/10.1371/journal.pone.0177643

- *Bat Mock Community COI* - A single marker example in bats, showing importance of the database content with the default classifier. Based on:

  > Walker *et al.* (2019) A fecal sequel: Testing the limits of a genetic assay for bat species identification. https://doi.org/10.1371/journal.pone.0224969

- *Synthetic controls with fungal ITS2* - A single marker example in fungi, with mock biological communities *and* synthetic control sequences. Based on:

Palmer *et al.* (2018) Non-biological synthetic spike-in controls and the AMPtk software pipeline improve mycobiome data. https://doi.org/10.7717/peerj.4925

- *Soil Nematode Mock Community* - Four markers (sequenced separately) in a soil nematode mock community. Based on:

  Ahmed *et al.* (2019) Metabarcoding of soil nematodes: the importance of taxonomic coverage and availability of reference sequences in choosing suitable marker(s) https://doi.org/10.3897/mbmg.3.36408

- *Pest Insect Mock Communities* - Three markers (sequenced together) in insect mock communities. Based on:

  Batovska *et al.* (2021) Developing a non-destructive metabarcoding protocol for detection of pest insects in bulk trap catches https://doi.org/10.1038/s41598-021-85855-6

- *Endangered Species Mixes 16S etc* - A dozen markers in animals and plants (sequenced together). Based on:

  Arulandhu *et al.* (2017) Development and validation of a multi-locus DNA metabarcoding method to identify endangered species in complex samples. https://doi.org/10.1093/gigascience/gix080

For each worked example there is a different sub-folder in the THAPBI PICT source code under `examples/` containing at least `setup.sh` to do one-off setup like downloading the public data, and `run.sh` to execute the main analysis discussed. There will usually be assorted other files like reference sequences, or `metadata.tsv`.

Running the examples will create or use subdirectories `raw_data/` for the downloaded FASTQ files, `intermediate/` for per-sample working files, and `summary/` for the final output reports. Where the example includes positive controls like mock communities, the expected species content is recorded under `expected/` in per-sample files.

CHAPTER

FIVE

REFERENCE DATABASE

5.1 Introduction

THAPBI PICT has been designed as a framework which can be applied to multiple biological contexts, demonstrated in the *worked examples*. Each new set of marker(s) (i.e. PCR primer targets) will require a new reference database be compiled, most likely starting from published sequences, but we also sequenced culture collections.

Applied to environmental samples, some primer pairs will amplify a much wider sequence space than others, either reflecting a more diverse genome region, or simply a longer sequence. Related to this, the fraction of observed sequences with a published reference will also vary - and thus the density of the references in sequence space. This in turn will can change which classifier algorithm is most appropriate. Inspecting the edit-graph produced for all your samples and your initial database entries can help interpret this.

The default classifier allows perfect matches, or a single base pair difference (substitution, insertion or deletion). This requires good database coverage with unambiguous sequences, which we have been able to achieve for the *Phytophthora* ITS1 region targeted by default.

5.2 Provided database

THAPBI provides a default database which is used when the command line `-d` or `--database` setting is omitted. This is intended for use with a *Phytophthora* ITS1 target region, and is used in the first *worked example*.

For further details see the `database/README.rst` file in the source code, and script `database/build_ITS1_DB.sh` which automates this.

5.3 Ambiguous bases in database

Ideally all the reference sequences in your database will have unambiguous sequences only (`A`, `C`, `G` and `T`). However, some published species sequences will contain IUPAC ambiguity codes, especially if capillary sequenced. How this is handled will depend on the classifier algorithm used.

For example *Phytophthora condilina* accession `KJ372262` has a single `W` meaning `A` or `T`. In this case for *P. condilina* in our curated set, we could select the unambiguous accession `MG707826` instead.

With the strictest `identity` classifier, the `W` will never be matched (since the Illumina platform does not produce any ambiguous bases other than `N`). With the default `onebp` classifier, this can match but the `W` would be the single allowed mismatch (and any database entry with more than one ambiguity would never be matched). The `blast` classifier uses NCBI BLAST+ internally, and would handle the base as expected.

5.4 Conflicting taxonomic assignments

With any amplicon marker, it is possible that distinct species will share the exact same sequence. For example, this happens with our ITS1 marker for model organism *Phytophthora infestans* and sister species *P. andina* and *P. ipomoeae*. In cases like this where the classifier finds multiple equally valid taxonomic assignments in the database, they are **all** reported. Should the user wish however, their database could record a single assignment like *Phytophthora infestans*-complex.

Our default primers for *Phytophthora* can amplify related genera, not just *Peronosporales*, but also some *Pythiales*. Expanding the database coverage runs into two main problems. First, with less published sequences available, the default strict classifier may fail to match many sequences to a published sequence. Second, with past renaming and splitting of some genera, the taxonomic annotation can becomes less consistent.

The `thapbi_pict conflicts` subcommand can be used to report any conflicts at species or genus level.

CHAPTER

SIX

ABUNDANCE & NEGATIVE CONTROLS

Any negative control sample is not expected to contain any of the target sequences (although it may contain spike-in synthetic control sequences).

On a typical 96-well plate of PCR products which will go on to be multiplexed for Illumina MiSeq sequencing, most of the samples are biological - but some should be negative controls (e.g. PCR blanks, or synthetic sequences). The presence of biological sequence reads in the negative control samples is indicative of some kind of cross contamination. Likewise, synthetic sequences in the biological samples are a warning sign.

The tool implements both absolute and fractional abundance thresholds, which can be specified at the command line. Moreover, control samples can be used to automatically raise the threshold for batches of samples. Simple negative controls can be used to set an absolute abundance threshold, but to set the fractional abundance threshold we need to be able to distinguish expected sequences from unwanted ones. For this we require known spike-in control sequences, which are clearly distinct from the biological markers.

6.1 Spike-in Controls

Four synthetic sequences were designed for Phyto-Threats project which funded THAPBI PICT. These were of the typical expected ITS1 fragment length and base content, had the typical fixed 32bp header, but were otherwise shuffled with no biological meaning (avoiding any secondary structure forming). They were synthesised using Integrated DNA Technologies gBlocks Gene Fragments.

Our 96-well PCR plates included multiple control samples which were known ratios of these synthetic sequences, rather than environmental DNA.

The tool needs a way to distinguish biological marker sequences (for which we wanted to make as few assumptions as possible) from the synthetic ones (where the template sequences were known, subject only to PCR noise).

The spike-in controls are assumed to be in the database, by default under the synthetic "genus" but that is configurable. Similar sequences in the samples are considered to be spike-ins. While the PCR noise is typically just a few base pair changes, we also found large deletions relatively common. The matching is therefore relaxed, currently based on *k*-mer content.

Conversely, the presence of the synthetic controls in any of the biological samples is also problematic. Since our synthetic control sequences are in the default database, they can be matched by the chosen classifier, and appear in the reports.

6.2 Minimum Absolute Abundance Threshold

The initial absolute abundance threshold is set at the command line with `-a` or `--abundance` giving an integer value. If your samples have dramatically different read coverage, then the fractional abundance threshold may be more appropriate (see below).

During the sample tally step, the `-n` or `--negctrls` argument gives the sample filenames of any negative controls to use to potentially increase the absolute abundance threshold (see below). If you have no spike-in controls, then any sequences in these negative controls can raise the threshold - regardless of what they may or may not match in the reference database.

6.3 Minimum Fractional Abundance Threshold

The initial fractional abundance threshold is set at the command line with `-f` or `--abundance-fraction` to an floating point number between zero and one, thus `-f 0.001` means 0.1%. This is a percentage of the reads identified for each marker after merging the overlapping pairs and primer matching.

During the read preparation step, the `-y` or `--synctrls` argument gives the sample filenames of any synthetic controls to use to potentially increase the absolute abundance threshold. This setting works in conjunction with the database which must include the spike-in sequences under the genera specified at the command lines with `--synthetic` (by default "synthetic").

6.4 Automatic thresholds

Any control samples are processed first, before the biological samples, and high read counts can raise the threshold to that level for the other samples in that folder. This assumes if you have multiple 96-well plates, or other logical groups, their raw FASTQ files are separated into a sub-folder per plate.

Control samples given via `-n` can raise the absolute abundance threshold (any synthetic spike-in reads are ignored for this), while controls given via `-y` can raise the fractional abundance threshold (but must have synthetic spike-in reads in order to give a meaningful fraction).

For example, if running with the default minimum abundance threshold of 100 (set via `-a 100`), and a negative control (set via `-n raw_data/CTRL*.fastq.gz`) contains a non-spike-in (and thus presumably biological) sequence at abundance 136, then the threshold for the non-control samples in that folder is raised to 136.

Alternatively, you might have synthetic spike-in controls listed with `-y raw_data/SPIKES*.fastq.gz` and use `-f 0.001` to set a default fractional abundance threshold of 0.1%. Suppose a control had 100,000 reads for a marker passing the overlap merging and primer matching, of which 99,800 matched the spike-ins leaving 200 unwanted presumably biological reads, of which the most abundant was at 176 copies. Then the fractional abundance threshold would be raised slightly to 176 / 100000 = 0.00176 or 0.176%.

Note that a control sample can be used with *both* `-n` and `-y`, so in this second example that would *also* raise the absolute abundance threshold to 176 reads.

Potentially a synthetic control sample can have unusually low read coverage, meaning even a low absolute number of non-spike-in reads (at noise level) would give a spuriously high inferred fractional abundance threshold. To guard against this corner case, as a heuristic half the absolute abundance threshold is applied to the synthetic control samples. Likewise, half of any fractional abundance threshold is applied to the negative control samples, which guards against spurious raising of the absolute threshold.

A similar problem would occur if you accidentally use `-y` on a sample without any expected spike-in controls. This would suggest result in an overly high fractional threshold, treated as an error.

CHAPTER

SEVEN

CLASSIFIER ASSESSMENT

In assessing classification performance, it is the combination of both classification method (algorithm) and marker database which which matters. Settings like the abundance threshold and any read correction are also important, and the tool default settings partly reflect one of the original project goals being to avoid false positives.

To objectively assess a metabarcoding classifier we require sequenced samples of known composition, which generally means single isolates (where a single marker sequence is typically expected), or mock communities (the bulk of our *worked examples*). Carefully controlled environmental samples are possible too. We use Muri *et al.* (2020) as a *worked example identifying fish species* where the lake was drained to collected and identify all the individual fish, but this is problematic as the lakes were large enough that DNA from each fish could not be expected at all the sampling points, giving an inflated false negative count.

Our tool includes an presence/absence based assessment framework based on supplying expected species lists for control samples, from which the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) counts can be computed for each species. These are the basis of standard metrics like sensitivity (recall), specificity, precision, F-score (F-measure, or F1), and Hamming Loss. It is simple but not overly helpful to apply metrics like this to each species, but the overall performance is more informative.

However, some scores like the Hamming Loss are fragile with regards to the TN count when comparing databases. The Hamming Loss given by the total number of mis-predicted class entries divided by the number of class-level predictions, thus (FP + FN) / (TP + FP + FN + TN). Consider a mock community of ten species, where the classifier made 11 predictions which break down as 9 TP and 2 FP, meaning 10 - 9 = 1 FN. Suppose the database had a hundred species (including all ten in the mock community), that leaves 100 - 9 - 1 - 2 = 88 TN, and a Hamming Loss of 3/100 = 0.03. Now suppose the database was extended with additional references not present in this mock community, perhaps expanding from European *Phytophthora* species to include distinct entries for tropical species, or a sister group like *Peronospora*. The denominator would increase, reducing the Hamming Loss, but intuitively the classifier performance on this mock community has not changed. To address this, the classifier assessment also includes a modified *ad-hoc* loss metric calculated as the total number of mis-predicted class entries divided by the number of class-level predictions ignoring TN, or (FP + FN) / (TP + FP + FN) which in this example would give 3/12 = 0.25 regardless of the number of species in the database. This is an intuitive measure weighting FP and FN equally (smaller is better, zero is perfect), a potential complement to the F-score.

Note that the assessment framework only considers species level predictions, ignoring genus only predictions, and thus will not distinguish between the default `onebp` classifier and variants like `1s3g`.

CHAPTER

EIGHT

COMMAND LINE

THAPBI PICT is a command line tool, meaning you must open your command line terminal window and key in instructions to use the tool. The documentation examples use the $ (dollar sign) to indicate the prompt, followed by text to be entered. For example, this should run the tool with no instructions:

```
$ thapbi_pict
...
```

Rather than literally printing dot dot dot, the tool should print out some terse help, listing various sub-command names, and an example of how to get more help.

For example, `-v` (minus sign, lower case letter v) or `--version` (minus, minus, version in lower case) can be added to find out the version of the tool installed:

```
$ thapbi_pict -v
THAPBI PICT v0.8.1
```

THAPBI PICT follows the sub-command style popularised in bioinformatics by `samtools` (also used in the version control tool `git`). This means most of the instructions take the form `thapbi_pict sub-command ...`, where the dots indicate some additional options.

The main sub-commands are to do with classifying sequence files and reporting the results, and these are described in the first *worked example*:

- `prepare` - turn paired FASTQ input files for each sample, giving de-duplicated FASTA files
- `fasta-nr` and `sample-tally` pooling intermediate files for analysis
- `classify` - produce genus/species level predictions as tab-separated-variable TSV files
- `summary` - summarise a set of predictions by sample (with human readable report), and by unique sequence and sample (both with Excel reports)
- `edit-graph` - draw the unique sequences as nodes on a graph, connected by edit-distance
- `assess` - compare classifier output to known positive controls
- `pipeline` - run all of the above in sequence

There are further sub-commands to do with making or inspecting an SQLite3 format barcode marker sequence database, most of which are covered in the second *worked example, with a custom database*:

- `dump` - export a DB as TSV or FASTA format
- `load-tax` - import a copy of the NCBI taxonomy
- `import` - import a FASTA file, e.g. using the NCBI style naming
- `conflicts` - report on genus or species level conflicts in the database

And some other miscellaneous commands:

- `ena-submit` - write a TSV table of your paired FASTQ files for use with the ENA interactive submission system.

Start with reading the help for any command using `-h` or `--help` as follows:

```
$ thapbi_pict pipeline -h
...
```

Most of the commands have required arguments, and if you omit a required argument it will stop with an error:

```
$ thapbi_pict pipeline
...
thapbi_pict pipeline: error: the following arguments are required: -i/--input, -o/--
↪output
```

CHAPTER

NINE

PYTHON API

THAPBI *Phytophthora* ITS1 Classifier Tool (PICT).

You would typically use THAPBI PICT via the command line tool it defines:

```
$ thapbi_pict --help
...
```

However, it is also possible to call functions etc from within Python. The top level package currently only defines the tool version:

```
>>> from thapbi_pict import __version__
>>> print(__version__)
```

The tool documentation is hosted by Read The Docs, generated automatically from the `docs/` folder of the software repository and the *"docstrings"* within the source code which document the Python API.

9.1 thapbi_pict.assess module

Assess classification of marker reads at species level.

This implements the `thapbi_pict assess ...` command.

`thapbi_pict.assess.`**`class_list_from_tally_and_db_list`**(*tally: dict[tuple[str, str], int]*, *db_sp_list: list[str]*) → list[str]

Sorted list of all class names used in a confusion table dict.

`thapbi_pict.assess.`**`extract_binary_tally`**(*class_name: str*, *tally: dict[tuple[str, str], int]*) → tuple[int, int, int, int]

Extract single-class TP, FP, FN, TN from multi-class confusion tally.

Reduces the mutli-class expectation/prediction to binary - did they include the class of interest, or not?

Returns a 4-tuple of values, True Positives (TP), False Positives (FP), False Negatives (FN), True Negatives (TN), which sum to the tally total.

`thapbi_pict.assess.`**`extract_global_tally`**(*tally: dict[tuple[str, str], int]*, *sp_list: list[str]*) → tuple[int, int, int, int]

Process multi-label confusion matrix (tally dict) to TP, FP, FN, TN.

If the input data has no negative controls, all there will be no true negatives (TN).

Returns a 4-tuple of values, True Positives (TP), False Positives (FP), False Negatives (FN), True Negatives (TN).

These values are analogous to the classical binary classifier approach, but are NOT the same. Even if applied to single class expected and predicted values, results differ:

- Expect none, predict none - 1xTN
- Expect none, predict A - 1xFP
- Expect A, predict none - 1xFN
- Expect A, predict A - 1xTP
- Expect A, predict B - 1xFP (the B), 1xFN (missing A)
- Expect A, predict A&B - 1xTP (the A), 1xFP (the B)
- Expect A&B, predict A&B - 2xTP
- Expect A&B, predict A - 1xTP, 1xFN (missing B)
- Expect A&B, predict A&C - 1xTP (the A), 1xFP (the C), 1xFN (missing B)

The TP, FP, FN, TN sum will exceed the tally total. For each tally entry, rather than one of TP, FP, FN, TN being incremented (weighted by the tally count), several can be increased.

If the input data has no negative controls, all there will be no TN.

thapbi_pict.assess.**load_tsv**(*mapping: dict[tuple[str, str], str]*, *classifier_file: str*, *min_abundance: int*) → dict[tuple[str, str], str]

Update dict mapping of (marker, MD5) to semi-colon separated species string.

thapbi_pict.assess.**main**(*inputs*, *known*, *db_url*, *method*, *min_abundance*, *assess_output*, *map_output*, *confusion_output*, *marker=None*, *ignore_prefixes=None*, *debug=False*)

Implement the (sample/species level) `thapbi_pict assess` command.

The inputs argument is a list of filenames and/or folders.

Must provide: * at least one XXX.<method>.tsv file * at least one XXX.<known>.tsv file

These files can cover multiple samples as the sample-tally based classifier output, or legacy per-sample <sample>.<known>.tsv files.

thapbi_pict.assess.**save_confusion_matrix**(*tally: dict[tuple[str, str], int]*, *db_sp_list: list[str]*, *sp_list: list[str]*, *filename: str*, *exp_total: int*, *debug: bool = False*) → None

Output a multi-class confusion matrix as a tab-separated table.

thapbi_pict.assess.**save_mapping**(*tally: dict[tuple[str, str], int]*, *filename: str*, *debug: bool = False*) → None

Output tally table of expected species to predicted sp.

thapbi_pict.assess.**sp_for_sample**(*fasta_files: list[str]*, *min_abundance: int*, *pooled_sp: dict[tuple[str, str], str]*) → str

Return semi-colon separated species string from FASTA files via dict.

thapbi_pict.assess.**sp_in_tsv**(*classifier_files: list[str]*, *min_abundance: int*) → str

Return semi-colon separated list of species in column 2.

Will ignore genus level predictions.

thapbi_pict.assess.**tally_files**(*expected_file: str*, *predicted_file: str*, *min_abundance: int = 0*) → dict[tuple[str, str], set[str]]

Make dictionary tally confusion matrix of species assignments.

Rather than the values simply being an integer count, they are the set of MD5 identifiers (take the length for the count).

9.2 thapbi_pict.classify module

Classifying prepared marker sequences using a marker database.

This implements the `thapbi_pict classify ...` command.

thapbi_pict.classify.**apply_method_to_seqs**(*method_fn: Callable*, *input_seqs: dict[str, str]*, *session*, *marker_name: str*, *min_abundance: int = 0*, *debug: bool = False*) → Iterator[tuple[str, str, str, str]]

Call given method on each sequence in the dict.

Assumes any abundance filter has already been applied. Input is a dict of identifiers mapped to upper case sequences.

thapbi_pict.classify.**consoliate_and_sort_taxonomy**(*genus_species_taxid: Iterable[tuple[str, str, int]]*) → list[tuple[str, str, int]]

Remove any redundant entries, returns new sorted list.

Drops zero taxid entries if has matching non-zero entry.

Drops genus only entries if have species level entries. Note ignoring the TaxID here - would need to know the parent/child relationship to confirm the genus we're removing does have species level children in the prediction set.

thapbi_pict.classify.**main**(*inputs: list[str]*, *session*, *marker_name: str*, *method: str*, *out_dir: str*, *ignore_prefixes: tuple[str]*, *tmp_dir: str*, *min_abundance: int = 0*, *biom=False*, *debug: bool = False*, *cpu: int = 0*) → list[str | None]

Implement the `thapbi_pict classify` command.

For use in the pipeline command, returns a filename list of the TSV classifier output.

The input files should have been prepared with the same or a lower minimum abundance - this acts as an additional filter useful if exploring the best threshold.

thapbi_pict.classify.**method_blast**(*input_seqs: dict[str, str]*, *session*, *marker_name: str*, *tmp_dir: str*, *shared_tmp_dir: str*, *min_abundance: int = 0*, *debug: bool = False*, *cpu: int = 0*) → Iterator[tuple[str, str, str, str]]

Classify using BLAST.

Another simplistic classifier, run the reads through blastn against a BLAST database of our marker sequence database entries.

thapbi_pict.classify.**method_cleanup**() → None

Free any memory and/or delete any files on disk.

Currently no need to generalise this for the different classifiers, but could if for example we also needed to delete any files on disk.

thapbi_pict.classify.**method_dist**(*input_seqs: dict[str, str]*, *session*, *marker_name: str*, *tmp_dir: str*, *shared_tmp_dir: str*, *min_abundance: int = 0*, *debug: bool = False*, *cpu: int = 0*) → Iterator[tuple[str, str, str, str]]

Classify using edit distance.

thapbi_pict.classify.**method_identity**(*input_seqs: dict[str, str]*, *session*, *marker_name: str*, *tmp_dir: str*, *shared_tmp_dir: str*, *min_abundance: int = 0*, *debug: bool = False*, *cpu: int = 0*) → Iterator[tuple[str, str, str, str]]

Classify using perfect identity.

This is a deliberately simple approach, in part for testing purposes. It looks for a perfect identical entry in the database.

thapbi_pict.classify.**method_substr**(*input_seqs: dict[str, str]*, *session*, *marker_name: str*, *tmp_dir: str*, *shared_tmp_dir: str*, *min_abundance: int = 0*, *debug: bool = False*, *cpu: int = 0*) → Iterator[tuple[str, str, str, str]]

Classify using perfect identity including as a sub-string.

Like the 'identity' method, but allows for a database where the marker has not been trimmed, or has been imperfectly trimmed (e.g. primer mismatch).

thapbi_pict.classify.**perfect_match_in_db**(*session*, *marker_name: str*, *seq: str*, *debug: bool = False*) → tuple[int | str, str, str]

Lookup sequence in DB, returns taxid, genus_species, note as tuple.

If the 100% matches in the DB give multiple species, then taxid and genus_species will be semi-colon separated strings.

thapbi_pict.classify.**perfect_substr_in_db**(*session*, *marker_name: str*, *seq: str*, *debug: bool = False*) → tuple[int | str, str, str]

Lookup sequence in DB, returns taxid, genus_species, note as tuple.

If the matches containing the sequence as a substring give multiple species, then taxid and genus_species will be semi-colon separated strings.

thapbi_pict.classify.**setup_blast**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*)

Prepare a BLAST DB from the marker sequence DB entries.

thapbi_pict.classify.**setup_dist2**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*) → None

Prepare a set of all DB marker sequences; set dist to 2.

thapbi_pict.classify.**setup_dist3**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*) → None

Prepare a set of all DB marker sequences; set dist to 3.

thapbi_pict.classify.**setup_dist4**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*) → None

Prepare a set of all DB marker sequences; set dist to 4.

thapbi_pict.classify.**setup_dist5**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*) → None

Prepare a set of all DB marker sequences; set dist to 5.

thapbi_pict.classify.**setup_dist6**(*session*, *marker_name*, *shared_tmp_dir*, *debug=False*, *cpu=0*)

Prepare a set of all DB marker sequences; set dist to 6.

thapbi_pict.classify.**setup_dist7**(*session*, *marker_name*, *shared_tmp_dir*, *debug=False*, *cpu=0*)

Prepare a set of all DB marker sequences; set dist to 7.

thapbi_pict.classify.**setup_dist8**(*session*, *marker_name*, *shared_tmp_dir*, *debug=False*, *cpu=0*)

Prepare a set of all DB marker sequences; set dist to 8.

thapbi_pict.classify.**setup_dist9**(*session*, *marker_name*, *shared_tmp_dir*, *debug=False*, *cpu=0*)

Prepare a set of all DB marker sequences; set dist to 9.

thapbi_pict.classify.**setup_onebp**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*) → None

Prepare a set of all the DB marker sequences; set dist to 1.

thapbi_pict.classify.**setup_seqs**(*session*, *marker_name: str*, *shared_tmp_dir: str*, *debug: bool = False*, *cpu: int = 0*) → None

Prepare a set of all the DB marker sequences as upper case strings.

Also setup set of sequences in the DB, and dict of genus to NCBI taxid.

thapbi_pict.classify.**taxid_and_sp_lists**(*taxon_entries: Iterable*) → tuple[int | str, str, str]

Return semi-colon separated summary of the taxonomy objects from DB.

Will discard genus level predictions (e.g. 'Phytophthora') if there is a species level prediciton within that genus (e.g. 'Phytophthora infestans').

If there is a single result, returns a tuple of taxid (integer), genus-species, and debugging comment (strings).

If any of the fields has conflicting values, returns two semi-colon separated string instead (in the same order so you can match taxid to species, sorting on the genus-species string).

thapbi_pict.classify.**unique_or_separated**(*values: Sequence[str | int]*, *sep: str = ';'*) → str

Return sole element, or a string joining all elements using the separator.

9.3 thapbi_pict.conflicts module

Explore conflicts at species and genus level.

thapbi_pict.conflicts.**main**(*db_url: str*, *output_filename: str*, *debug: bool = False*) → int

Implement the `thapbi_pict conflicts` subcommand.

Looks for taxonomy conflicts at marker, genus or species level, with the number of marker or genus level conflicts used as the return code. i.e. Unix failure (non-zero) when there are marker or genus level conflicts.

A marker level conflict is when a unique sequence appears in the DB under more than one marker name (e.g. both COI and ITS1), which is most likely an error in the DB construction.

Genus level conflicts are where a unique sequence in the DB is reported from more than one genus, which is considered undesirable. Similarly for species level conflicts, but for some markers this is sadly common and not considered to be an error.

9.4 thapbi_pict.denoise module

Apply UNOISE read-correction to denoise FASTA file(s).

This implements the `thapbi_pict denoise ...` command, which is a simplified version of the `thapbi_pict sample-tally ...` command intended to be easier to use outside the THAPBI PICT pipeline.

thapbi_pict.denoise.**main**(*inputs: str | list[str]*, *output: str*, *denoise_algorithm: str*, *total_min_abundance: int = 0*, *min_length: int = 0*, *max_length: int = 9223372036854775807*, *unoise_alpha: float | None = None*, *unoise_gamma: int | None = None*, *gzipped: bool = False*, *tmp_dir: str | None = None*, *debug: bool = False*, *cpu: int = 0*)

Implement the `thapbi_pict denoise` command.

This is a simplified version of the `thapbi_pict sample-tally` command which pools one or more FASTA input files before running the UNOISE read correction algorithm to denoise the dataset. The input sequences should use the SWARM <prefix>_<abundance> style naming, which is used on output (taking the first loaded name if a sequence appears more than once).

Arguments min_length and max_length are applied while loading the input FASTA file(s).

Argument total_min_abundance is applied after read correction.

Results sorted by decreasing abundance, then alphabetically by sequence.

thapbi_pict.denoise.**read_correction**(*algorithm: str*, *counts: dict[str, int]*, *unoise_alpha: float | None = 2.0*, *unoise_gamma: int | None = 4*, *abundance_based: bool = False*, *tmp_dir: str | None = None*, *debug: bool = False*, *cpu: int = 0*) → tuple[dict[str, str], dict[str, str]]

Apply builtin UNOISE algorithm or invoke an external tool like VSEARCH.

Argument algorithm is a string, "unoise-l" for our reimplementation of the UNOISE2 algorithm, or "usearch" or "vsearch" to invoke those tools at the command line.

Argument counts is an (unsorted) dict of sequences (for the same amplicon marker) as keys, with their total abundance counts as values.

Returns a dict mapping input sequences to centroid sequences, and dict of any chimeras detected (empty for some algorithms).

thapbi_pict.denoise.**unoise**(*counts: dict[str, int]*, *unoise_alpha: float | None = 2.0*, *unoise_gamma: int | None = 4*, *abundance_based: bool = False*, *debug: bool = False*) → tuple[dict[str, str], dict[str, str]]

Apply UNOISE2 algorithm.

Argument counts is an (unsorted) dict of sequences (for the same amplicon marker) as keys, with their total abundance counts as values.

If not specified (i.e. set to zero or None), unoise_alpha defaults to 2.0 and unoise_gamma to 4.

Returns a dict mapping input sequences to centroid sequences, and an empty dict (no chimera detection performed).

thapbi_pict.denoise.**usearch**(*counts: dict[str, int]*, *unoise_alpha: float | None = None*, *unoise_gamma: int | None = None*, *abundance_based: bool = False*, *tmp_dir: str | None = None*, *debug: bool = False*, *cpu: int = 0*) → tuple[dict[str, str], dict[str, str]]

Invoke USEARCH to run its implementation of the UNOISE3 algorithm.

Assumes v10 or v11 (or later if the command line API is the same). Parses the four columns tabbed output.

Returns a dict mapping input sequences to centroid sequences, and a dict of MD5 checksums of any sequences flagged as chimeras.

thapbi_pict.denoise.**vsearch**(*counts: dict[str, int]*, *unoise_alpha: float | None = None*, *unoise_gamma: int | None = None*, *abundance_based: bool = True*, *tmp_dir: str | None = None*, *debug: bool = False*, *cpu: int = 0*) → tuple[dict[str, str], dict[str, str]]

Invoke VSEARCH to run its reimplementation of the UNOISE3 algorithm.

Argument counts is an (unsorted) dict of sequences (for the same amplicon marker) as keys, with their total abundance counts as values.

Returns a dict mapping input sequences to centroid sequences, and a dict of MD5 checksums of any sequences flagged as chimeras.

9.5 thapbi_pict.db_import module

Shared code for THAPBI PICT to import FASTA into our database.

This code is used for importing NCBI formatted FASTA files, our curated ITS1 sequence FASTA file databases, and other other FASTA naming conventions.

thapbi_pict.db_import.**import_fasta_file**(*fasta_file*, *db_url*, *fasta_entry_fn*, *entry_taxonomy_fn*, *marker*, *left_primer=None*, *right_primer=None*, *min_length=None*, *max_length=None*, *name=None*, *trim=True*, *debug=True*, *validate_species=False*, *genus_only=False*, *tmp_dir=None*)

Import a FASTA file into the database.

thapbi_pict.db_import.**load_taxonomy**(*session*) → set[str]

Pre-load all the species and synonym names as a set.

thapbi_pict.db_import.**lookup_genus**(*session*, *name: str*)

Find genus entry via taxonomy/synonym table (if present).

thapbi_pict.db_import.**lookup_species**(*session*, *name: str*)

Find this species entry in the taxonomy/synonym table (if present).

thapbi_pict.db_import.**main**(*fasta*, *db_url*, *marker*, *left_primer=None*, *right_primer=None*, *min_length=0*, *max_length=9223372036854775807*, *name=None*, *convention='simple'*, *sep=None*, *validate_species=False*, *genus_only=False*, *ignore_prefixes=None*, *tmp_dir=None*, *debug=False*)

Import FASTA file(s) into the database.

For curated FASTA files, use convention "simple" (default here and at the command line), and specify any multi-entry separator you are using.

For NCBI files, convention "ncbi" and for the separator use Ctrl+A (type `-s $'\001'` at the command line) if appropriate, or "" or None (function default) if single entries are expected.

thapbi_pict.db_import.**parse_curated_fasta_entry**(*text: str*, *known_species: list[str] | None = None*) → tuple[int, str]

Split an entry of "Accession genus species etc" into fields.

Does not use the optional known_species argument.

Returns a two-tuple of taxid (0 unless taxid=… entry found), genus-species.

```
>>> parse_curated_fasta_entry("HQ013219 Phytophthora arenaria")
(0, 'Phytophthora arenaria')
```

Will look for an NCBI taxid after the species name (and ignore anything following that, such as other key=value entries):

```
>>> parse_curated_fasta_entry("P13660 Phytophthora aff infestans taxid=907744 etc")
(907744, 'Phytophthora aff infestans')
```

In this example we expect the NCBI taxid will be matched to a pre-loaded species name to be used in preference (i.e. 'Phytophthora aff. infestans' with a dot in it).

thapbi_pict.db_import.**parse_ncbi_fasta_entry**(*text: str*, *known_species: list[str] | None = None*) → tuple[int, str]

Split an entry of Accession Genus Species-name Description.

Returns a two-tuple: taxid (always zero), presumed genus-species (may be the empty string).

```
>>> parse_ncbi_fasta_entry("LC159493.1 Phytophthora drechsleri genes ...")
(0, 'Phytophthora drechsleri')
>>> parse_ncbi_fasta_entry("A57915.1 Sequence 20 from Patent EP0751227")
(0, '')
>>> parse_ncbi_fasta_entry("Y08654.1 P.cambivora ribosomal internal ...")
(0, '')
```

If a list of known species are used, then right most word is dropped until the text matches a known name. This discards any description (and strain level information if the list is only to species level).

If there is no match to the provided names, heuristics are used but this defaults to the first two words.

Dividing the species name into genus, species, strain etc is not handled here.

thapbi_pict.db_import.**parse_ncbi_taxid_entry**(*text: str*, *know_species: list[str] | None = None*) → tuple[int, str]

Find any NCBI taxid as a pattern in the text.

Returns a two-tuple of taxid (zero if not found), and an empty string (use the taxonomy table in the DB to get the genus-species).

Uses a regular expression based on taxid=<digits>, and only considers the first match:

```
>>> parse_ncbi_taxid_entry("HQ013219 Phytophthora arenaria [taxid=]")
(0, '')
>>> parse_ncbi_taxid_entry("HQ013219 Phytophthora arenaria [taxid=123] [taxid=456]")
(123, '')
```

thapbi_pict.db_import.**parse_obitools_fasta_entry**(*text: str*, *known_species: list[str] | None = None*) → tuple[int, str]

Parse species from the OBITools extended FASTA header.

See https://pythonhosted.org/OBITools/attributes.html which explains that OBITools splits the FASTA line into identifier, zero or more key=value; entries, and a free text description.

We are specifically interested in the species_name, genus_name (used if species_name is missing), and taxid.

```
>>> entry = "AP009202 species_name=Abalistes stellaris; taxid=392897; ..."
>>> parse_obitools_fasta_entry(entry)
(392897, 'Abalistes stellaris')
```

Note this will *not* try to parse any key=value entries embedded in the first word (which taken as the identifier).

thapbi_pict.db_import.**parse_sintax_fasta_entry**(*text: str*, *known_species: list[str] | None = None*) → tuple[int, str]

Extract the species from SINTAX taxonomy annotation.

See https://drive5.com/usearch/manual/tax_annot.html which defines this taxonomy annotation convention as used in USEARCH and VSEARCH. The tax=names field is separated from other fields in the FASTA description line by semi-colons, for example:

```
>>> entry = "X80725_S000004313;tax=d:...,g:Escherichia/Shigella,s:Escherichia_coli"
>>> parse_sintax_fasta_entry(entry)
(0, 'Escherichia coli')
```

If there is no species entry (prefix `s:`) then the genus is returned (prefix `g:`), else the empty string:

```
>>> parse_sintax_fasta_entry("AB008314;tax=d:...,g:Streptococcus;")
(0, 'Streptococcus')
```

If the species entry is missing the genus information (which may happen depending how the file was generated), that is inferred heuristically:

```
>>> entry = "X80725_S000004313;tax=d:...,g:Escherichia,s:coli"
>>> parse_sintax_fasta_entry(entry)
(0, 'Escherichia coli')
```

This can be unclear:

```
>>> entry = ">X80725_S000004313;tax=d:...,g:Escherichia/Shigella,s:Escherichia_coli"
>>> parse_sintax_fasta_entry(entry)
(0, 'Escherichia coli')
```

9.6 thapbi_pict.db_orm module

Object Relational Mapping for marker sequence database.

Using SQLalchemy, the Python classes defined here give us a database schema and the code to import/export the data as Python objects.

class thapbi_pict.db_orm.**Base**(**args: Any*, ***kwargs: Any*)

Bases: DeclarativeBase

Base class for SQLAlchemy ORM declarations.

See the SQLAlchemy 2.0 documentation. This is expected to be compatible with type checkers like mypy.

class thapbi_pict.db_orm.**DataSource**(**args: Any*, ***kwargs: Any*)

Bases: *Base*

Database entry for a data source (NCBI, curated, etc).

Each accession is expected to be unique within a data source.

class thapbi_pict.db_orm.**MarkerDef**(**args: Any*, ***kwargs: Any*)

Bases: *Base*

Database entry for a marker listing primers and amplicon length limits.

class thapbi_pict.db_orm.**MarkerSeq**(**args: Any*, ***kwargs: Any*)

Bases: *Base*

Database entry for a single marker reference sequence.

class thapbi_pict.db_orm.**SeqSource**(**args: Any*, ***kwargs: Any*)

Bases: *Base*

Database entry for source of a marker sequence entry.

marker_definition

alias of *MarkerDef*

marker_seq

alias of *MarkerSeq*

source

alias of *DataSource*

taxonomy

alias of *Taxonomy*

class thapbi_pict.db_orm.**Synonym**(**args: Any*, ***kwargs: Any*)

Bases: *Base*

Database entry for a synonym of a taxonomy entry.

In addition to direct synonyms, includes the names and synonyms of any child nodes of the species (e.g. variants, strains, etc).

class thapbi_pict.db_orm.**Taxonomy**(**args: Any*, ***kwargs: Any*)

Bases: *Base*

Database entry for a species' taxonomy entry.

thapbi_pict.db_orm.**connect_to_db**(**args*, ***kwargs*)

Create engine and return session bound to it.

```
>>> Session = connect_to_db("sqlite:///:memory:", echo=True)
>>> session = Session()
```

9.7 thapbi_pict.dump module

Dumping out marker database to text files.

This implements the `thapbi_pict dump ...` command.

thapbi_pict.dump.**main**(*db_url: str*, *output_filename: str*, *output_format: str*, *marker: str | None = None*, *minimal: bool = False*, *genus: str = ''*, *species: str = ''*, *sep: str | None = None*, *debug: bool = True*)

Run the database dump with arguments from the command line.

thapbi_pict.dump.**none_str**(*value*, *none_value: str = ''*) → str

Turn value into a string, special case None to empty string.

9.8 thapbi_pict.edit_graph module

Generate edit-distance network graph from FASTA files.

This implements the `thapbi_pict edit-graph ...` command.

thapbi_pict.edit_graph.**main**(*graph_output: str*, *graph_format: str*, *db_url: str*, *input_file: str*, *min_abundance: int = 100*, *show_db_marker: str | None = None*, *total_min_abundance: int = 0*, *min_samples: int = 0*, *max_edit_dist: int = 3*, *ignore_prefixes: tuple[str, ...] | None = None*, *debug: bool = False*) → int

Run the edit-graph command with arguments from the command line.

This shows sequences from a database (possibly filtered with species/genus limits) and/or selected sample-tally TSV file (optionally with classifier output, and possibly with a minimum abundance limit set here).

Computes a Levenshtein edit-distance matrix from the selected sequences, which can be exported as a matrix, but is usually converted into a graph of unique sequences as nodes, with short edit distances as edges.

Graph node size is scaled by sample count (number of FASTA files that it appears in), and colored by assigned species (from a classifier TSV file).

thapbi_pict.edit_graph.**write_pdf**(*G*, *handle*) → None

Render NetworkX graph to PDF using GraphViz fdp.

thapbi_pict.edit_graph.**write_xgmml**(*G*, *handle*, *name: str = 'THAPBI PICT edit-graph'*) → None

Save graph in XGMML format suitable for Cytoscape import.

9.9 thapbi_pict.fasta_nr module

Prepare a non-redundant FASTA file using MD5 naming.

This implements the `thapbi_pict fasta-nr ...` command, using some of the same code internally as the `thapbi_pict prepare-reads` command.

thapbi_pict.fasta_nr.**main**(*inputs: str | list[str]*, *revcomp: str | list[str]*, *output: str*, *min_abundance: int = 0*, *min_length: int = 0*, *max_length: int = 9223372036854775807*, *debug: bool = False*) → None

Implement the `thapbi_pict fasta-nr` command.

9.10 thapbi_pict.sample_tally module

Prepare a non-redundant TSV file using MD5 naming.

This implements the `thapbi_pict sample-tally ...` command.

thapbi_pict.sample_tally.**main**(*inputs: str | list[str]*, *synthetic_controls: list[str]*, *negative_controls: list[str]*, *output: str*, *session*, *marker: str | None = None*, *spike_genus=None*, *fasta=None*, *min_abundance: int = 100*, *min_abundance_fraction: float = 0.001*, *total_min_abundance: int = 0*, *min_length: int = 0*, *max_length: int = 9223372036854775807*, *denoise_algorithm: str = '-'*, *unoise_alpha: float = 2.0*, *unoise_gamma: int = 4*, *gzipped: bool = False*, *biom: str | None = None*, *tmp_dir: str | None = None*, *debug: bool = False*, *cpu: int = 0*) → None

Implement the `thapbi_pict sample-tally` command.

Arguments min_length and max_length are applied while loading the input per-sample FASTA files.

Argument algorithm is a string, "-" for no read correction (denoising), "unoise-l" for our reimplementation of the UNOISE2 algorithm, or "usearch" or "vsearch" to invoke those tools at the command line.

Arguments min_abundance and min_abundance_fraction are applied per-sample (after denoising if being used), increased by pool if negative or synthetic controls are given respectively. Comma separated string argument spike_genus is treated case insensitively.

9.11 thapbi_pict.prepare module

Prepare raw amplicon sequencing reads (trimming, merging, etc).

This implements the `thapbi_pict prepare-reads ...` command.

thapbi_pict.prepare.**find_fastq_pairs**(*filenames_or_folders: list[str]*, *ext: tuple[str, ...] = ('.fastq', '.fastq.gz', '.fq', '.fq.gz')*, *ignore_prefixes: tuple[str] | None = None*, *debug: bool = False*) → list[tuple[str, str, str]]

Interpret a list of filenames and/or foldernames.

Returns a list of tuples (stem, left filename, right filename) where stem is intended for use in logging and output naming, and may include a directory name.

The filenames will be normalised relative to the current directory (so that we can directly compare file lists which could have been defined inconsistently by the user).

Will ignore "-" if present in the inputs.

thapbi_pict.prepare.**load_marker_defs**(*session*, *spike_genus: str = ''*) → dict[str, dict[str, int | str | list[tuple[str, str, set[str]]]]]

Load marker definitions and any spike-in sequences from the DB.

thapbi_pict.prepare.**main**(*fastq: list[str]*, *out_dir: str*, *session*, *flip: bool = False*, *min_abundance: int = 2*, *min_abundance_fraction: float = 0.0*, *ignore_prefixes: tuple[str] | None = None*, *merged_cache: str | None = None*, *tmp_dir: str | None = None*, *debug: bool = False*, *cpu: int = 0*) → list[str]

Implement the `thapbi_pict prepare-reads` command.

For use in the pipeline command, returns a filename listing of the FASTA files created.

thapbi_pict.prepare.**make_nr_fasta**(*input_fasta_or_fastq: str*, *output_fasta: str*, *min_abundance: int = 0*, *min_len: int = 0*, *max_len: int = 9223372036854775807*, *weighted_input: bool = False*, *fastq: bool = False*, *gzipped: bool = False*, *header_dict: dict[str, str | int | None] | None = None*, *debug: bool = False*) → tuple[int, int, int, int]

Trim and make non-redundant FASTA/Q file from FASTA input.

Makes a non-redundant FASTA file with the sequences named `>MD5_abundance\n`.

For FASTQ files all input reads are treated as abundance one (using weighted_input=True gives an error).

If FASTA input and weighted_input=True, reads must follow `>identifier_abundance\n` naming and the abundance is used. Otherwise all treated as abundance one.

Makes a non-redundant FASTA file with the sequences named `>MD5_abundance\n`.

Returns the total number of accepted reads before de-duplication (integer), number of those unique (integer), and the total number of those which passed the minimum abundance threshold (integer), and number of those which are unique (integer).

thapbi_pict.prepare.**marker_cut**(*marker_definitions*, *file_pairs: list[tuple[str, str, str]]*, *out_dir: str*, *merged_cache: str*, *tmp: str*, *flip: bool*, *min_abundance: int*, *min_abundance_fraction: float*, *debug: bool = False*, *cpu: int = 0*) → list[str]

Apply primer-trimming for given markers.

thapbi_pict.prepare.**merge_paired_reads**(*raw_R1: str*, *raw_R2: str*, *merged_fasta_gz: str*, *tmp: str*, *debug: bool = False*, *cpu: int = 0*) → tuple[int, int]

Create NR FASTA file by overlap merging the paired FASTQ files.

thapbi_pict.prepare.**parse_cutadapt_stdout**(*stdout: str*) → tuple[int, int]

Extract FASTA count before and after cutadapt.

```
>>> parse_cutadapt_stdout(
...     "...\n"
...     "Total reads processed: 5,869\n"
...     "...\n"
...     "Reads written (passing filters): 5,861 (99.9%)\n"
...     "..."
... )
(5869, 5861)
```

thapbi_pict.prepare.**parse_flash_stdout**(*stdout: str*) → tuple[int, int]

Extract FASTQ pair count before/after running flash.

```
>>> parse_flash_stdout(
...     "...\n"
...     "[FLASH] Read combination statistics:[FLASH]     Total pairs:      6105\n"
...     "[FLASH]     Combined pairs:   5869\n"
...     "..."
... )
(6105, 5869)
```

thapbi_pict.prepare.**prepare_sample**(*fasta_name: str*, *trimmed_fasta: str*, *headers: dict[str, int | str | None]*, *min_len: int*, *max_len: int*, *min_abundance: int*, *min_abundance_fraction: float*, *tmp: str*, *debug: bool = False*, *cpu: int = 0*) → tuple[int | None, int | None, int | None, int]

Create marker-specific FASTA file for sample from paired FASTQ.

Applies abundance threshold, and min/max length.

Returns pre-threshold total read count, accepted unique sequence count, accepted total read count, and the absolute abundance threshold used (higher of the given absolute threshold or the given fractional threshold).

thapbi_pict.prepare.**run_cutadapt**(*long_in: str*, *out_template: str*, *marker_definitions: dict[str, Any]*, *flip: bool = False*, *debug: bool = False*, *cpu: int = 0*) → tuple[int, int]

Run cutadapt on a single file (i.e. after merging paired FASTQ).

The input and/or output files may be compressed as long as they have an appropriate suffix (e.g. gzipped with `.gz` suffix).

Returns FASTA count before and after cutadapt.

thapbi_pict.prepare.**run_flash**(*trimmed_R1: str*, *trimmed_R2: str*, *output_dir: str*, *output_prefix: str*, *debug: bool = False*, *cpu: int = 0*) → tuple[int, int]

Run FLASH on a pair of trimmed FASTQ files to merge overlapping pairs.

Returns two integers, FASTQ pair count for input and output files.

thapbi_pict.prepare.**save_nr_fasta**(*counts: dict[str, int]*, *output_fasta: str*, *min_abundance: int = 0*, *gzipped: bool = False*, *header_dict: dict[str, str | int | None] | None = None*) → tuple[int, int]

Save a dictionary of sequences and counts as a FASTA file.

Writes a FASTA file with header lines starting # (which not all tools will accept as valid FASTA format).

The output FASTA records are named `>MD5_abundance\n`, which is the default style used in SWARM. This could in future be generalised, for example `>MD5;size=abundance;\n` for the VSEARCH default.

Results are sorted by decreasing abundance then alphabetically by sequence.

Returns the total and number of unique sequences accepted (above any minimum abundance specified).

Use output_fasta='-' for standard out.

9.12 thapbi_pict.summary module

Summarise classification results at sample and read level.

This implements the `thapbi_pict summary ...` command.

The code uses the term metadata to refer to the user-provided information about each sample (via a plain text TSV table), and statistics for the internally tracked information about each sample like the number of raw reads in the original FASTQ files (via header lines in the intermediate FASTA files).

thapbi_pict.summary.**color_bands**(*meta_groups*, *sample_color_bands*, *default_fmt=None*, *debug: bool = False*) → list

Return a list for formats, one for each sample.

thapbi_pict.summary.**main**(*inputs*, *report_stem: str*, *method: str*, *min_abundance: int = 1*, *metadata_file: str | None = None*, *metadata_encoding: str | None = None*, *metadata_cols: str | None = None*, *metadata_groups: str | None = None*, *metadata_fieldnames: str | None = None*, *metadata_index: str | None = None*, *require_metadata: bool = False*, *show_unsequenced: bool = True*, *ignore_prefixes: tuple[str] | None = None*, *biom: bool = False*, *debug: bool = False*) → int

Implement the `thapbi_pict summary` command.

The expectation is that the inputs represent all the samples from a meaningful group, likely from multiple sequencing runs (plates).

thapbi_pict.summary.**read_summary**(*markers*, *marker_md5_to_seq*, *marker_md5_species*, *marker_md5_abundance*, *abundance_by_samples*, *stem_to_meta*, *meta_names*, *group_col*, *sample_stats*, *stats_fields*, *output*, *method*, *min_abundance=1*, *excel=None*, *biom=None*, *debug=False*) → None

Create reads (rows) vs species (cols) report.

The expectation is that the inputs represent all the samples from one (96 well) plate, or some other meaningful batch.

thapbi_pict.summary.**sample_summary**(*sample_species_counts*, *meta_to_stem*, *stem_to_meta*, *meta_names*, *group_col*, *sample_stats*, *stats_fields*, *show_unsequenced*, *output*, *excel*, *method*, *min_abundance=1*, *debug=False*)

Create samples (rows) vs species (cols) report.

The expectation is that the inputs represent all the samples from a meaningful group, likely from multiple sequencing runs (plates).

9.13 thapbi_pict.taxdump module

Code for THAPBI PICT to deal with NCBI taxonomy dumps.

The code is needed initially for loading an NCBI taxdump folder (files `names.dmp`, `nodes.dmp`, `merged.dmp` etc) into a marker database.

thapbi_pict.taxdump.**filter_tree**(*tree: dict[int, int]*, *ranks: dict[str, set[int]]*, *ancestors: set[int]*) → tuple[dict[int, int], dict[str, set[int]]]

Return a filtered version of the tree & ranks dict.

NOTE: Does NOT preserve the original dict order.

thapbi_pict.taxdump.**get_ancestor**(*taxid: int*, *tree: dict[int, int]*, *stop_nodes: set[int]*) → int

Walk up tree until reach a stop node, or root.

thapbi_pict.taxdump.**load_merged**(*merged_dmp: str*, *wanted: set[int] | None = None*) → dict[int, int]

Load mapping of merged taxids of interest from NCBI taxdump merged.dmp file.

thapbi_pict.taxdump.**load_names**(*names_dmp: str*, *wanted: set[int] | None = None*) → tuple[dict[int, str], dict[int, set[str]]]

Load scientific names of species from NCBI taxdump names.dmp file.

thapbi_pict.taxdump.**load_nodes**(*nodes_dmp: str*, *wanted_ranks: Sequence[str] | None = None*) → tuple[dict[int, int], dict[str, set[int]]]

Load the NCBI taxdump nodes.dmp file.

Returns two dicts, the parent/child relationships, and the ranks (values are lists of taxids).

Default is all ranks, can provide a possibly empty list/set of ranks of interest.

thapbi_pict.taxdump.**main**(*tax: str*, *db_url: str*, *ancestors: str*, *debug: bool = True*) → int

Load an NCBI taxdump into a database.

thapbi_pict.taxdump.**not_top_species**(*tree: dict[int, int]*, *ranks: dict[str, set[int]]*, *names: dict[int, str]*, *synonyms: dict[int, set[str]]*, *top_species*) → Iterator[tuple[int, str]]

Find all 'minor' species, takes set of species taxid to ignore.

Will map assorted sub-species (i.e. any nodes under top_species) to the parent species, e.g. varietas 'Phytophthora nicotianae var. parasitica' NCBI:txid4791 will be mapped to species 'Phytophthora nicotianae' NCBI:txid4790 instead.

Will map anything else to the parent genus, although generally it will be skipped via the reject_species_name(...) function, e.g.

- no-rank entry 'unclassified Pythium' NCBI:txid228096 would be mapped to Pythium NCBI:txid4797 - although we'd not interested in importing any unclassified entries.
- no-rank entry 'environmental samples' NCBI:txid660914 would be mapped to genus 'Hyaloperonospora' NCBI:txid184462 - but we skip this.
- entry 'uncultured Hyaloperonospora' NCBI:txid660915 would be mapped to genus 'Hyaloperonospora' NCBI:txid184462 - but we skip uncultured.

However, if you wanted to import this part of the tree:

- clade entry 'Skeletonema marinoi-dohrnii complex' NCBI:txid1171708 would be mapped to genus 'Skeletonema' NCBI:txid2842

Yields (genus taxid, node name) tuples.

thapbi_pict.taxdump.**species_or_species_groups**(*tree: dict[int, int]*, *ranks: dict[str, set[int]]*, *names: dict[int, str]*) → Iterator[tuple[int, int]]

Find taxids for species or species groups.

Our "genus" list matches the NCBI rank "genus", and includes child nodes as aliases (unless they fall on our "species" list or reject list of "environmental samples" or "unclassified <genus>").

However, our "species" list are either NCBI rank "species" or "species group" (in the later case child species are taken as aliases).

Does not distinguish between "top level" species, or those under "no rank" nodes like "environmental samples" or "unclassified Phytophthora" (taxid 211524),

Yields (species taxid, genus taxid) tuples.

9.14 thapbi_pict.utils module

Helper functions for THAPB-PICT code.

thapbi_pict.utils.**abundance_filter_fasta**(*input_fasta: str*, *output_fasta: str*, *min_abundance: int*) → None

Apply a minimum abundance filter to a FASTA file.

thapbi_pict.utils.**abundance_from_read_name**(*text: str*, *debug: bool = False*) → int

Extract abundance from SWARM style read name.

```
>>> abundance_from_read_name("9e8f051c64c2b9cc3b6fcb27559418ca_988")
988
```

If fails, will return one.

thapbi_pict.utils.**abundance_values_in_fasta**(*fasta_file: str*, *gzipped: bool = False*) → tuple[int, int, dict[str, int]]

Return unique count, total abundance, and maximum abundances by spike-in.

thapbi_pict.utils.**cmd_as_string**(*cmd*)

Express a list command as a suitably quoted string.

Intended for using in debugging or error messages.

thapbi_pict.utils.**expand_IUPAC_ambiguity_codes**(*seq: str*)

Convert to upper case and iterate over possible unabmigous interpretations.

This is a crude recursive implementation, intended for use on sequences with just a few ambiguity codes in them - it may not scale very well!

thapbi_pict.utils.**export_sample_biom**(*output_file: str*, *seqs: dict[tuple[str, str], str]*, *seq_meta: dict[tuple[str, str], dict]*, *sample_meta: dict[str, dict[str, str | int | None]]*, *counts: dict[tuple[str, str, str], int]*, *gzipped: bool = True*) → bool

Export a sequence vs samples counts BIOM table, with metadata.

Similar to the export_sample_tsv file (our TSV output), expects same arguments as loaded from one of our TSV files via the parse_sample_tsv function.

Will save a BIOM v2 HDF5 file if possible and return True. If output fails (e.g. cannot import the biom Python library), returns False.

thapbi_pict.utils.**export_sample_tsv**(*output_file: str*, *seqs: dict[tuple[str, str], str]*, *seq_meta: dict[tuple[str, str], dict]*, *sample_meta: dict[str, dict[str, str]]*, *counts: dict[tuple[str, str, str], int]*, *gzipped: bool = False*) → None

Export a sequence vs sample counts TSV table, with metadata.

The TSV file ought to be readable by the parse_sample_tsv function, and is first generated in our pipeline by the sample-tally command, and then extended by the classify command to add taxonomic sequence metadata.

If the output tabular file argument is "-", it writes to stdout (not supported with gzipped mode).

With no sequence metadata this should be accepted as a TSV BIOM file.

thapbi_pict.utils.**file_to_sample_name**(*filename: str*) → str

Given filename (with or without a directory name), return sample name only.

i.e. XXX.fasta, XXX.fastq.gz, XXX.method.tsv –> XXX

thapbi_pict.utils.**find_paired_files**(*filenames_or_folders*, *ext1*, *ext2*, *ignore_prefixes=None*, *debug=False*, *strict=False*)

Interpret a list of filenames and/or foldernames to find pairs.

Looks for paired files named XXX.ext1 and XXX.ext2 which can be in different directories - duplicated filenames (in different directories) are considered to be an error.

Having XXX.ext1 without XXX.ext2 is an error in strict mode, or a warning in debug mode, otherwise silently ignored.

Having XXX.ext2 without XXX.ext1 is silently ignored.

The arguments ext1 and ext2 should include the leading dot.

thapbi_pict.utils.**find_requested_files**(*filenames_or_folders: list[str]*, *ext: str | tuple[str, ...] = '.fasta'*, *ignore_prefixes: tuple[str] | None = None*, *debug: bool = False*) → list[str]

Interpret a list of filenames and/or foldernames.

The extensions argument can be a tuple.

thapbi_pict.utils.**genus_species_name**(*genus: str*, *species: str*) → str

Return name, genus with species if present.

Copes with species being None (or empty string).

thapbi_pict.utils.**genus_species_split**(*name: str*) → tuple[str, str]

Return (genus, species) splitting on first space.

If there are no spaces, returns (name, '') instead.

thapbi_pict.utils.**is_spike_in**(*sequence: str*, *spikes: list[tuple[str, str, set[str]]]*) → str

Return spike-in name if sequence matches, else empty string.

thapbi_pict.utils.**iskeyword**()

x.__contains__(y) <==> y in x.

thapbi_pict.utils.**kmers**(*sequence: str*, *k: int = 31*) → set[str]

Make set of all kmers in the given sequence.

thapbi_pict.utils.**load_fasta_header**(*fasta_file*, *gzipped=False*) → dict

Parse our FASTA hash-comment line header as a dict.

thapbi_pict.utils.**load_metadata**(*metadata_file*, *metadata_encoding*, *metadata_cols*, *metadata_groups=None*, *metadata_name_row=1*, *metadata_index=0*, *metadata_index_sep=';'*, *ignore_prefixes=('Undetermined',)*, *debug=False*)

Load specified metadata as several lists.

The encoding argument can be None or "", meaning use the default.

The columns argument should be a string like "1,3,5" - a comma separated list of columns to output. The column numbers are assumed to be one-based as provided by the command line user.

The name row indicates which row in the table contains the names or descriptions of the metadata columns (one-based).

The index column is assumed to contain one or more sequenced sample names separated by the character specified (default is semi-colon). This one-to-many mapping reflecting that a single field sample could be sequenced more than once (e.g. technical replicates). These sample names are matched against the file name stems, see function find_metadata.

The metadata table rows are sorted based on the requested colunms.

Return values:

- Dict mapping FASTQ stems to metadata tuples
- Ordered dict mapping metadata tuples to lists of FASTQ stems
- list of the N field names
- Color grouping offset into the N values

thapbi_pict.utils.**md5_hexdigest**(*filename: str*, *chunk_size: int = 1024*) → str

Return the MD5 hex-digest of the given file.

thapbi_pict.utils.**md5seq**(*seq: str*) → str

Return MD5 32-letter hex digest of the (upper case) sequence.

```
>>> md5seq("ACGT")
'f1f8f4bf413b16ad135722aa4591043e'
```

thapbi_pict.utils.**onebp_deletions**(*seq: str*) → set[str]

Generate all variants of sequence with 1bp deletion.

Assumes unambiguous IUPAC codes A, C, G, T only.

thapbi_pict.utils.**onebp_inserts**(*seq: str*) → set[str]

Generate all variants of sequence with 1bp insert.

Assumes unambiguous IUPAC codes A, C, G, T only.

thapbi_pict.utils.**onebp_substitutions**(*seq: str*)

Generate all 1bp substitutions of the sequence.

Assumes unambiguous IUPAC codes A, C, G, T only.

thapbi_pict.utils.**parse_sample_tsv**(*tabular_file: str*, *min_abundance: int = 0*, *debug: bool = False*, *force_upper: bool = True*) → tuple[dict[tuple[str, str], str], dict[tuple[str, str], dict[str, str]], dict[str, dict[str, str]], dict[tuple[str, str, str], int]]

Parse file of sample abundances and sequence (etc).

Optional argument min_abundance is applied to the per sequence per sample values (i.e. the matrix elements, not the row/column totals).

Columns are: * Sequence label, <marker>/<identifier>_<abundance> * Column per sample giving the sequence count * Sequence itself * Optional additional columns for sequence metadata (e.g. chimera flags)

Supports optional sample metadata header too as # prefixed header lines.

Returns dictionaries of: * Sequence keyed on [<marker>, <identitifer>], string * Sequence metadata keyed [<marker>, <identitifer>], dict of key:value pairs * Sample metadata keyed on [<sample>], dict of key:value pairs * Counts keyed on 3-tuple [<marker>, <identifier>, <sample>], integer

thapbi_pict.utils.**parse_species_tsv**(*tabular_file*, *min_abundance=0*, *req_species_level=False*, *allow_wildcard=False*) → Iterator[tuple[str | None, str, str, str]]

Parse file of species assignments/predictions by sequence.

Yields tuples of marker name (from the file header line), sequence name, taxid, and genus_species.

thapbi_pict.utils.**primer_clean**(*primer: str*) → str

Handle non-IUPAC entries in primers, maps I for inosine to N.

```
>>> primer_clean("I")
'N'
```

Inosine is found naturally at the wobble position of tRNA, and can match any base. Structurally similar to guanine (G), it preferentially binds cytosine (C). It sometimes used in primer design (Ben-Dov et al, 2006), where degeneracy N would give similar results.

thapbi_pict.utils.**reject_species_name**(*species: str*) → bool

Reject species names like 'environmental samples' or 'uncultured ...'.

Will also reject names with ";" in them as used in the classifier and reports to combine multiple species entries.

thapbi_pict.utils.**run**(*cmd*, *debug: bool = False*, *attempts: int = 1*) → CompletedProcess

Run a command via subprocess, abort if fails.

Returns a subprocess.CompletedProcess object, or None if all attempts fail.

thapbi_pict.utils.**species_level**(*prediction: str*) → bool

Is this prediction at species level.

Returns True for a binomial name (at least one space), False for genus only or no prediction.

thapbi_pict.utils.**split_read_name_abundance**(*text: str*, *debug: bool = False*) → tuple[str, int]

Split SWARM style read name into prefix and abundance.

```
>>> abundance_from_read_name("9e8f051c64c2b9cc3b6fcb27559418ca_988")
'9e8f051c64c2b9cc3b6fcb27559418ca', 988
```

If fails to detect the abundance, will return the original text as the prefix with an abundance of 1.

thapbi_pict.utils.**valid_marker_name**(*text: str*) → bool

Check the proposed string valid for use as a marker name.

Want to be able to use the string for file or directory names, and also column names etc in reports. At very least should reject whitespace, line breaks, and slashes.

Also rejecting all digits, as might want to accept integers as argument (e.g. cluster array job mapping task numbers to marker numbers).

Also rejecting the underscore, as may want to use it as a field separator in sequence names (e.g. `marker_md5_abundance`), and full stop as may use it as a field separator in filenames.

May want to relax this later, thus defining this central function.

9.15 thapbi_pict.ena_submit module

Code for sample submission to ENA/SRA.

This implements the `thapbi_pict ena-submit ...` command.

`thapbi_pict.ena_submit.`**`load_md5`**(*file_list: list[str]*) → dict[str, str]

Return a dict mapping given filenames to MD5 digests.

`thapbi_pict.ena_submit.`**`main`**(*fastq: list[str]*, *output: str*, *metadata_file: str | None = None*, *metadata_encoding: str | None = None*, *metadata_cols: str | None = None*, *metadata_fieldnames: str | None = None*, *metadata_index: str | None = None*, *ignore_prefixes: str | None = None*, *library_name: str = '-'*, *instrument_model: str = 'Illumina MiSeq'*, *design_description: str = ''*, *library_construction_protocol: str = ''*, *insert_size: int = 250*, *tmp_dir: str | None = None*, *debug: bool = False*)

Implement the `thapbi_pict ena-submit` command.

`thapbi_pict.ena_submit.`**`write_table`**(*handle*, *pairs: list[tuple[str, str, str]]*, *meta: dict[str, str] | None*, *library_name: str*, *instrument_model: str*, *design_description: str*, *library_construction_protocol: str*, *insert_size: int*) → None

Write read file table for ENA upload.

9.16 thapbi_pict.versions module

Helper code to get command line tool versions.

Defines various functions to check a tool is on the `$PATH` and if so, return the tool version as a short string (sometimes including a date).

These functions are called from various THAPBI-PICT subcommands which call external tools to ensure a clear missing dependency message, and to log the version of the external tool used.

If the tool is not on the path, the commands all return None.

If we cannot parse the output, again the commands return None - which is likely an indication of a major version change, meaning the tool ought to be re-evaluated for use with THAPBI-PICT.

`thapbi_pict.versions.`**`check_rapidfuzz`**() → str

Check can import rapidfuzz and confirm recent enough.

`thapbi_pict.versions.`**`check_tools`**(*names: list[str]*, *debug: bool*) → list[str]

Verify the named tools are present, log versions if debug=True.

Argument names should be an interable of tool binary names.

If all the tools are present, returns a list of version strings.

If any tools are missing (or have a version we could not parse), aborts.

thapbi_pict.versions.**version_blast**(*cmd: str = 'blastn'*) → str | None

Return the version of the NCBI BLAST+ suite's blastn (as a short string).

In the absence of a built in version switch like `-v`, this works by parsing the short help output with `-h` (which does vary between the tools in the suite):

```
$ makeblastdb -h | grep BLAST
Application to create BLAST databases, version 2.7.1+

$ blastn -h | grep BLAST
Nucleotide-Nucleotide BLAST 2.7.1+
```

In the above examples, it would behave as follows:

```
>>> version_blast("makeblastdb")
'2.7.1+'
>>> version_blast("blastn")
'2.7.1+'
```

If the command is not on the path, returns None.

thapbi_pict.versions.**version_cutadapt**(*cmd: str = 'cutadapt'*) → str | None

Return the version of cutadapt (as a short string).

Uses the output with `--version`:

```
$ cutadapt --version
1.18
```

It would capture this:

```
>>> version_cutadapt()
'1.18'
```

If the command is not on the path, returns None.

thapbi_pict.versions.**version_flash**(*cmd: str = 'flash'*) → str | None

Return the version of flash (as a short string).

Parses the output with `-v`:

```
$ flash -v | head -n 1
FLASH v1.2.11
```

It would capture the version from the first line as follows:

```
>>> version_flash()
'v1.2.11'
```

If the command is not on the path, returns None.

thapbi_pict.versions.**version_graphviz_fdp**(*cmd: str = 'fdp'*) → str | None

Return the version of the GraphViz tool fdp (as a short string).

Depends on the -V switch:

```
$ fdp -V
fdp - graphviz version 9.0.0 (0)
```

In the above example, it would behave as follows:

```
>>> version_graphviz_fdp()
'9.0.0'
```

If the command is not on the path, returns None.

thapbi_pict.versions.**version_usearch**(*cmd: str = 'usearch'*) → str | None

Return the version of usearch (as a short string).

Uses the output with `--version`:

```
$ usearch --version
usearch v11.0.667_i86linux32
```

It would capture this:

```
>>> version_vsearch()
'v11.0.667'
```

If the command is not on the path, returns None.

thapbi_pict.versions.**version_vsearch**(*cmd: str = 'vsearch'*) → str | None

Return the version of vsearch (as a short string).

Uses the output with `--version`:

```
$ vsearch --version
...
vsearch v2.22.1_macos_x86_64, 8.0GB RAM, 8 cores
...
```

It would capture this:

```
>>> version_vsearch()
'v2.22.1'
```

If the command is not on the path, returns None.

CHAPTER

TEN

RELEASE HISTORY

Version	Date	Notes
v1.0.12	2024-03-11	Restored Python 3.8 support. More robust import of SINTAX style FASTA files.
v1.0.11	2024-03-05	Harmonize ASV naming in BIOM output, optional `sample-tally` BIOM output.
v1.0.10	2024-02-26	Sample report 'Unique' column is now the unique ASV count. Misc updates.
v1.0.9	2024-02-12	Using Python type annotations (internal code change). Python 3.9 onwards.
v1.0.8	2024-02-06	Additional curated *Phytophthora* in default DB. Adds `1s6g` classifier.
v1.0.7	2024-01-29	Treat *Phytophthora cambivora* as a synonym of *Phytophthora x cambivora*.
v1.0.6	2024-01-24	Added some *Peronosclerospora* to curated DB. Updated NCBI import.
v1.0.5	2023-11-22	Updated NCBI import, and scripted most of what was a semi-manual process.
v1.0.4	2023-11-20	Dropped unused `-m` / `--method` argument to `edit-graph` command.
v1.0.3	2023-09-04	Updated NCBI import and curated *P. condilina* entries in default DB.
v1.0.2	2023-08-18	Use sum of cutadapt and singleton values etc for pooled marker reports.
v1.0.1	2023-07-26	Fixed some rare corner-case read-corrections in `unoise-l` mode.
v1.0.0	2023-05-19	Minor documentation changes, linked to Cock *et al.* (2023) preprint.
v0.14.1	2023-03-13	Optional BIOM output using the `biom-format` Python library.
v0.14.0	2023-03-02	Offers UNOISE read-correction, built-in or invoking USEARCH or VSEARCH.
v0.13.6	2022-12-28	Factional abundance threshold in `sample-tally` was not strict enough.
v0.13.5	2022-12-21	Misc small fixes and documentation updates.
v0.13.4	2022-12-07	Support abundance thresholding in the `sample-tally` step. Log controls.
v0.13.3	2022-11-25	Using new `sample-tally` command in pipeline, not `fasta-nr`.
v0.13.2	2022-11-11	Sped up `substr` classifier, especially with larger databases.
v0.13.1	2022-09-21	Minor default DB update. Cap `--cpu` by available CPUs. Faster DB import.
v0.13.0	2022-09-14	Sped up distance based classifiers by better use of RapidFuzz library.
v0.12.9	2022-08-19	Updates default DB with new curated species and improved left trimming.
v0.12.8	2022-08-08	Treat NCBI taxonomy 'equivalent name' as a synonym. Minor DB update.
v0.12.7	2022-07-26	NCBI taxid in genus-only fallback classifier output. Minor DB update.
v0.12.6	2022-07-25	Changes to how NCBI sequences are trimmed for use in the default DB.
v0.12.5	2022-07-08	Merged/child NCBI taxid entries as synonyms. Import FASTA with taxid.
v0.12.4	2022-07-07	Updated edit-graph code to work with RapidFuzz v2.0.0 or later.
v0.12.3	2022-07-06	Updated NCBI taxonomy and bulk genus-only entries in default DB.
v0.12.2	2022-06-15	Updates to the curated entries in the default *Phytophthora* ITS1 DB.
v0.12.1	2022-05-18	Fix missing field regression on reports including unsequenced samples.
v0.12.0	2022-04-19	Set fractional abundance threshold via synthetic spike-ins. Cutadapt v4.0+.
v0.11.6	2022-03-09	Fix regression on reports including unsequenced samples.
v0.11.5	2022-02-18	Reporting enhancements when using spike-in (synthetic) controls.
v0.11.4	2022-02-08	Updates to default curated DB, adding several more *Phytophthora* species.
v0.11.3	2022-02-01	Fix dynamic *k*-mer threshold for synthetic spike-in control sequences.
v0.11.2	2022-01-20	Windows testing on AppVeyor, with minor Windows specific fixes.
v0.11.1	2022-01-18	Using `rapidfuzz` rather than `python-Levenshtein`.

continues on next page

Table 1 – continued from previous page

Version	Date	Notes
v0.11.0	2022-01-13	Multi-marker reports, pooling predictions from each marker.
v0.10.6	2022-01-12	Fixed slow-down in v0.10.0 on large datasets with small DB.
v0.10.5	2021-12-23	Default for `-f` / `--abundance-fraction` is now 0.001, meaning 0.1%.
v0.10.4	2021-11-24	Updates to default curated DB, including newer NCBI taxonomy.
v0.10.3	2021-11-19	New `-f` / `--abundance-fraction` setting, off by default.
v0.10.2	2021-11-05	Updates to default curated DB. Small changes to NCBI taxonomy loading.
v0.10.1	2021-07-28	Fix for using SQLAlchemy v1.3 (previous release needed v1.4).
v0.10.0	2021-07-28	Rework to handle larger DB and multiple markers. Modifies DB schema.
v0.9.9	2021-07-08	Drop SWARM based classifiers. Single intermediate TSV file in pipeline.
v0.9.8	2021-06-17	Drop edit-graph in pipeline. Require full length primers in merged reads.
v0.9.7	2021-06-04	USEARCH SINTAX & OBITools FASTA conventions in `import` command.
v0.9.6	2021-05-21	Update default DB taxonomy, Peronosporales & Pythiales max 450bp.
v0.9.5	2021-05-10	Simplify to just one `import` command for pre-trimmed FASTA input.
v0.9.4	2021-05-05	Drop unused metadata fields in DB schema. Fix GML format edit graphs.
v0.9.3	2021-05-04	Drop HMM for spike-in control detection, now via DB & *k*-mer counting.
v0.9.2	2021-04-28	Fix obscure problem using relative versions of absolute paths.
v0.9.1	2021-04-20	Set metadata encoding. Spike-in HMM default now off.
v0.9.0	2021-04-19	Drop use of Trimmomatic, faster and slightly higher read counts.
v0.8.4	2021-04-13	Sped up re-running by delaying method setup until and if required.
v0.8.3	2021-04-13	Include abundance threshold in summary reports (if varied by sample).
v0.8.2	2021-04-13	Sample report pooling script. Fix `-p` in `prepare-reads`.
v0.8.1	2021-04-09	Drop species list embedded in intermediate TSV, `assess` needs DB now.
v0.8.0	2021-04-06	Revise genus/species columns in sample report. Add `scripts/` folder.
v0.7.11	2021-03-30	`assess` now only at sample level. Abundance threshold in `classify`.
v0.7.10	2021-03-24	Pipeline includes `fasta-nr` command making non-redundant FASTA file.
v0.7.9	2021-03-15	Option to show unsequenced entries in summary sample report (`-u`).
v0.7.8	2021-03-11	Only import IUPAC DNA characters to DB. Fix *N. valdiviana* in default DB.
v0.7.7	2021-02-24	Revise default ITS1 DB: NCBI Oomycetes, more curation & single isolates.
v0.7.6	2021-02-17	`curated-seq` replaces `seq-import`, used when building default DB.
v0.7.5	2021-02-16	Refine default DB by adjusting how genus-level NCBI import trimmed.
v0.7.4	2021-02-15	Edit-graph genus-only labels. New `1s2g`, `1s4g` & `1s5g` classifiers.
v0.7.3	2021-01-29	Update NCBI import, taxonomy. New `1s3g` classifier. Use cutadapt v3.0+.
v0.7.2	2020-10-06	New `ena-submit` command for use with interactive ENA read submission.
v0.7.1	2020-09-29	Curated *Phytophthora* DB minor updates. Classifier output in edit-graph.
v0.7.0	2020-04-02	Read counts etc as a header in intermediate FASTA files; shown in reports.
v0.6.15	2020-03-12	Fix regression in read report column sorting.
v0.6.14	2020-03-12	Merge `read-summary` & `sample-summary` into new `summary` command.
v0.6.13	2020-03-09	New classifier method `substr` for poorly trimmed DB content.
v0.6.12	2020-03-09	New advanced setting `--merged-cache` intended for multiple marker use.
v0.6.11	2020-03-02	Update genus-level only NCBI import, restrict to those with 32bp leader.
v0.6.10	2020-02-24	Treat I (for inosine as in tRNA) in primers as N (IUPAC code for any base).
v0.6.9	2020-02-20	Allow pre-primer-trimmed FASTQ. Fix row coloring when missing samples.
v0.6.8	2020-02-17	Metadata `-x` default now column 1. Fix read report metadata captions.
v0.6.7	2020-02-13	Method in `pipeline` filenames; max sample abundance in read reports.
v0.6.6	2020-02-05	Coloring groups in `sample-report`. Can call assessment from `pipeline`.
v0.6.5	2020-01-27	Do `--flip` in `prepare-reads` using cutadapt v2.8 or later.
v0.6.4	2020-01-23	`curated-import` accepts primers. Reduce memory usage for `onebp`.
v0.6.3	2020-01-20	Treat NCBI taxonomy "includes" as synonyms, 396 new species aliases.
v0.6.2	2020-01-14	Memory optimisation to the default `onebp` classifier.
v0.6.1	2020-01-08	Requires at least Python 3.6 as now using f-strings (internal change only).

continues on next page

Table 1 – continued from previous page

Version	Date	Notes
v0.6.0	2020-01-08	Stop discarding normally conserved *Phytophthora* ITS1 marker 32bp start.
v0.5.8	2019-12-11	Correction to start of a *P. parsiana* curated sequence in our DB.
v0.5.7	2019-12-09	Replace min bit score with min percentage coverage in `blast` classifier.
v0.5.6	2019-12-04	Import species under "unclassified *Phytophthora*" as genus *Phytophthora*.
v0.5.5	2019-12-03	Update NCBI taxonomy, adds *Phytophthora caryae* and *P. pseudopolonica*.
v0.5.4	2019-12-02	Only use HMM to detect synthetic read negative controls.
v0.5.3	2019-11-25	Replace HMM filter on importing to the database with length check only.
v0.5.2	2019-11-25	Remove redundant use of HMM filter in `seq-import` command.
v0.5.1	2019-11-22	Update NCBI taxonomy, adds *Phytophthora oreophila* and *P. cacuminis*.
v0.5.0	2019-11-21	Only use HMM as a filter, not for trimming in DB import or classify steps.
v0.4.19	2019-11-19	Additional curated entries in default ITS1 database.
v0.4.18	2019-11-19	Rework `sample-summary` table output, now samples vs species with Excel.
v0.4.17	2019-11-15	Control based minimum abundance threshold applied at folders level.
v0.4.16	2019-11-15	Bug fix in `fasta-nr` when using input records with descriptions.
v0.4.15	2019-11-04	Harmonise `dump` FASTA & `curated-import` with semi-colon separator.
v0.4.14	2019-10-23	Configurable FASTA entry separator for `curated-import` & `ncbi-import`.
v0.4.13	2019-10-22	Fix 5 cases missing `A` near end, `...CTGAAAACT` to `...CTGAAAAACT`.
v0.4.12	2019-10-22	Remove now unused `legacy-import` and `database/legacy/` files.
v0.4.11	2019-10-21	Update curated DB entries, focused on truncated sequences.
v0.4.10	2019-10-21	New `curated-import` command, rework handling of curated DB entries.
v0.4.9	2019-10-17	New `sample-summary` switch `-q` / `--requiremeta`. NetworkX v2.4 fix.
v0.4.8	2019-10-11	New `fasta-nr` command for use in alternatives to `prepare-reads`.
v0.4.7	2019-10-10	New `--minlen` & `--maxlen` args for `prepare-reads` and `pipeline`.
v0.4.6	2019-10-02	Forgot to include updated DB with the PyPI release.
v0.4.5	2019-10-02	Apply primer trimming to `ncbi-import` (crop if primers found).
v0.4.4	2019-10-02	New `--hmm` & `--flip` arguments for `prepare-reads` and `pipeline`.
v0.4.3	2019-09-26	New `conflicts` command reports genus/species level conflicts in DB.
v0.4.2	2019-09-26	Drop clade from taxonomy table, require unique species entries.
v0.4.1	2019-09-16	Include NCBI strains/variants/etc & their synonyms as species synonyms.
v0.4.0	2019-09-12	NCBI taxonomy synonym support; *Oomycetes* default taxonomy import.
v0.3.12	2019-09-12	New `dump` option `-m` / `--minimal` for DB comparison.
v0.3.11	2019-09-09	Update default DB and tests to use September 2019 NCBI taxonomy.
v0.3.10	2019-09-05	Handle missing or empty input FASTQ files more gracefully.
v0.3.9	2019-08-14	Log BLAST bit score, merge `assess` warnings, 3dp for ad-hoc loss.
v0.3.8	2019-08-09	The `blast` classifier now applies a minimum BLAST bit score of 100.
v0.3.7	2019-08-05	Add Python API to the main documentation.
v0.3.6	2019-07-19	Add Zenodo FASTQ link to worked example and use `assess` command.
v0.3.5	2019-07-12	Add missing `T` or `CT` to 11 of the legacy ITS1 sequences in the DB.
v0.3.4	2019-07-08	Worked example using woody hosts dataset from Riddell *et al.* (2019).
v0.3.3	2019-07-04	Fix regression in group coloring for `read-summary` Excel output.
v0.3.2	2019-07-04	Read The Docs; use `-i` / `--input` consistently - no positional args.
v0.3.1	2019-06-27	Reformat documentation to use reStructuredText rather than Markdown.
v0.3.0	2019-06-26	Include four gBlocks synthetic negative controls in DB and pipeline.
v0.2.6	2019-06-25	*Phytophthora* ITS1 HMM threshold set within model file, not in code.
v0.2.5	2019-06-21	Include XGMML edit-graph (for Cytoscape use) in `pipeline` output.
v0.2.4	2019-06-21	Fix 3 *Hyaloperonospora* also in *Peronospora* in default DB.
v0.2.3	2019-06-18	Sample count rather than total read abundance for edit-graph node size.
v0.2.2	2019-06-12	New `edit-graph` command. Use Cytoscape etc, or PDF via GraphViz.
v0.2.1	2019-05-27	Cope better with multiple (short) ITS1 fragments during classification.
v0.2.0	2019-05-14	Limit ITS1 length, 100-250bp. Exclude uncultured NCBI entries from DB.

continues on next page

Table 1 – continued from previous page

Version	Date	Notes
v0.1.12	2019-05-09	Sort `read-summary` by species. Set coloring group at command line.
v0.1.11	2019-05-06	Excel output from `read-summary` with formatting applied.
v0.1.10	2019-05-03	Tweak command line API, renamed `plate-summary` to `read-summary`.
v0.1.9	2019-05-02	New `pipeline` subcommand (prepare reads, classify, and report).
v0.1.8	2019-05-01	Standard errors for missing external tools. Log versions in verbose mode.
v0.1.7	2019-05-01	Chang default classifier method from `identity` to more fuzzy `onebp`.
v0.1.6	2019-04-30	Include ready to use binary ITS1 DB in source tar-ball & wheel files.
v0.1.5	2019-04-29	Rework optional metadata integration and its display in summary reports.
v0.1.4	2019-04-25	Sort samples using the optional metadata fields requested in reports.
v0.1.3	2019-04-24	Can optionally display sample metadata from TSV file in summary reports.
v0.1.2	2019-04-17	Keep searching if `onebp` classifier perfect match is at genus-level only.
v0.1.1	2019-04-16	Expand default taxonomy & DB from Peronosporaceae to Peronosporales.
v0.1.0	2019-04-04	Include a bundled ITS1 DB.
v0.0.15	2019-04-03	Support for genus-level only entries in the DB.
v0.0.14	2019-04-01	MD5 in dump output. Fix importing sequences failing taxonomic validation.
v0.0.13	2019-03-22	Drop conserved 32bp when primer trim. Assess at sample level by default.
v0.0.12	2019-03-11	Fix bug in `swarmid` classifier.
v0.0.11	2019-03-08	Sped up FASTQ preparation by using `flash` instead of `pear` v0.9.6.
v0.0.10	2019-03-06	Replace primer code allowing only 1bp differences with `cutadapt`.
v0.0.9	2019-03-05	Look for expected primers, discards mismatches. Cache HMM files locally.
v0.0.8	2019-02-21	Fix multi-class TN under-counting. New loss metric, `swarmid` classifier.
v0.0.7	2019-02-12	New `plate-summary` command, `onebp` classifier.
v0.0.6	2019-02-07	Misc. cleanup and import fixes.
v0.0.5	2019-02-06	Hamming Loss in assessment output.
v0.0.4	2019-01-24	New `seq-import` command, `blast` classifier, multi-taxon predictions.
v0.0.3	2019-01-22	Simplify generated filenames.
v0.0.2	2019-01-21	New `assess` command.
v0.0.1	2019-01-17	Initial framework with `identity` and `swarm` classifiers.

CHAPTER

ELEVEN

DEVELOPMENT NOTES

11.1 Python style conventions

The Python code follows PEP8 and PEP257 docstring style, guided by the Zen of Python.

Practically, coding style is enforced with several command line tools including ruff (which implements a faster version of black) and flake8 (with plugins) run via the tool pre-commit.

You can install these tools using:

```
$ pip install pre-commit
$ pre-commit install  # within the thapbi_pict main directory
```

The checks will then run automatically when you make a git commit. You can also run the checks directly using:

```
$ pre-commit run -a
```

If your editor can be configured to run flake8 and/or ruff automatically, even better. These checks are done as part of the continuous integration when changes are made on GitHub.

11.2 Continuous Integration

Currently this is setup to do automated testing under Linux using free continuous integration services:

- CircleCI (Linux): https://circleci.com/gh/peterjc/thapbi-pict/tree/master
- AppVeyor (Windows): https://ci.appveyor.com/project/peterjc/thapbi-pict/history

11.3 Dependencies

See the main installation instructions for end users. For development we need Python, a bash shell, git, and various other command line dependencies. Installing THAPBI PICT from source (see below), will fetch Python dependencies.

The two requirements files (`requirements.txt` for Python dependencies, and `requirements-ext.txt` for external command line bioinformatics tools) are used in the continuous integration testing. These files can contain exact pinned dependency versions, allowing us to define a more reproducible environment for running this software if needed.

On Linux or macOS, you should have the bash shell and standard Unix tools like `grep` already installed. We recommend installing our specific command line tool dependencies with Conda packaging system, via the BioConda channel:

```
$ conda install --file requirements-ext.txt
```

On Windows, few of the dependencies are available via Conda. The Git For Windows installer will provide `git`, `bash`, `grep`, etc. You will also need to manually install sqlite3, flash, and NCBI BLAST.

11.4 Installing from source

First, download the code from GitHub and decompress it if required. The best way to do this if you are likely to contribute any changes is at the command line with `git`.

```
$ git clone https://github.com/peterjc/thapbi-pict.git
$ cd thapbi-pict
```

Then build the default reference database, by loading the provided FASTA files into SQLite3, see `database/README.rst` for more information on this. Make it read only to prevent accidental edits:

```
$ cd database
$ ./build_ITS1_DB.sh
$ cd ..
$ cp database/ITS1_DB.sqlite thapbi_pict/ITS1_DB.sqlite
$ chmod a-w thapbi_pict/ITS1_DB.sqlite
```

Assuming your default Python is at least version 3.7, to install the tool and automatically get our Python dependencies:

```
$ pip install .
```

If your system defaults to Python 2, try `pip3 install .` or `python3 -m pip install .` instead.

Once installed, you should be able to run the tool using:

```
$ thapbi_pict
```

This should automatically find the installed copy of the Python code. Use `thapbi_pict -v` to report the version, or `thapbi_pict -h` for help.

11.5 Release process

For a release, start from a clean git checkout (to reduce the chance of bundling any stray local files despite a cautious `MANIFEST.in`). You will need some python tools:

```
$ pip install -U pip twine build
```

First confirm if the DB at `thapbi_pict/ITS1_DB.sqlite` is up to date:

```
sqlite3 thapbi_pict/ITS1_DB.sqlite .dump | grep -i "Imported with" | head -n 1
```

If there have been changes requiring the DB be rebuilt, do this:

```
cd database
./build_ITS1_DB.sh
git commit ITS1_DB.fasta -m "Rebuilt DB"
cd ..
```

Next confirm the `CHANGELOG.rst` file is up to date, including using today's date for the new version. Then actually do the build:

```
rm -rf build/
python -m build
git tag vX.Y.Z
git push origin master --tags
twine upload dist/thapbi_pict-X.Y.Z*
```

The PyPI upload should trigger an automated pull request updating the THAPBI PICT BioConda recipe which will need reviewing (e.g. new dependencies) before it is merged.

Must also turn the git tag into a "release" on GitHub, and attach the wheel to it. This will generate a version specific DOI on Zenodo. https://github.com/peterjc/thapbi-pict/releases

This documentation was generated for THAPBI PICT version 1.0.12.

PYTHON MODULE INDEX

t

thapbi_pict, 143
thapbi_pict.assess, 143
thapbi_pict.classify, 145
thapbi_pict.conflicts, 147
thapbi_pict.db_import, 149
thapbi_pict.db_orm, 151
thapbi_pict.denoise, 147
thapbi_pict.dump, 152
thapbi_pict.edit_graph, 152
thapbi_pict.ena_submit, 162
thapbi_pict.fasta_nr, 153
thapbi_pict.prepare, 154
thapbi_pict.sample_tally, 153
thapbi_pict.summary, 156
thapbi_pict.taxdump, 157
thapbi_pict.utils, 158
thapbi_pict.versions, 162

INDEX

A

abundance_filter_fasta() (*in module thapbi_pict.utils*), 158
abundance_from_read_name() (*in module thapbi_pict.utils*), 158
abundance_values_in_fasta() (*in module thapbi_pict.utils*), 158
apply_method_to_seqs() (*in module thapbi_pict.classify*), 145

B

Base (*class in thapbi_pict.db_orm*), 151

C

check_rapidfuzz() (*in module thapbi_pict.versions*), 162
check_tools() (*in module thapbi_pict.versions*), 162
class_list_from_tally_and_db_list() (*in module thapbi_pict.assess*), 143
cmd_as_string() (*in module thapbi_pict.utils*), 158
color_bands() (*in module thapbi_pict.summary*), 156
connect_to_db() (*in module thapbi_pict.db_orm*), 152
consoliate_and_sort_taxonomy() (*in module thapbi_pict.classify*), 145

D

DataSource (*class in thapbi_pict.db_orm*), 151

E

expand_IUPAC_ambiguity_codes() (*in module thapbi_pict.utils*), 158
export_sample_biom() (*in module thapbi_pict.utils*), 158
export_sample_tsv() (*in module thapbi_pict.utils*), 158
extract_binary_tally() (*in module thapbi_pict.assess*), 143
extract_global_tally() (*in module thapbi_pict.assess*), 143

F

file_to_sample_name() (*in module thapbi_pict.utils*), 159
filter_tree() (*in module thapbi_pict.taxdump*), 157
find_fastq_pairs() (*in module thapbi_pict.prepare*), 154
find_paired_files() (*in module thapbi_pict.utils*), 159
find_requested_files() (*in module thapbi_pict.utils*), 159

G

genus_species_name() (*in module thapbi_pict.utils*), 159
genus_species_split() (*in module thapbi_pict.utils*), 159
get_ancestor() (*in module thapbi_pict.taxdump*), 157

I

import_fasta_file() (*in module thapbi_pict.db_import*), 149
is_spike_in() (*in module thapbi_pict.utils*), 159
iskeyword() (*in module thapbi_pict.utils*), 159

K

kmers() (*in module thapbi_pict.utils*), 159

L

load_fasta_header() (*in module thapbi_pict.utils*), 159
load_marker_defs() (*in module thapbi_pict.prepare*), 154
load_md5() (*in module thapbi_pict.ena_submit*), 162
load_merged() (*in module thapbi_pict.taxdump*), 157
load_metadata() (*in module thapbi_pict.utils*), 159
load_names() (*in module thapbi_pict.taxdump*), 157
load_nodes() (*in module thapbi_pict.taxdump*), 157
load_taxonomy() (*in module thapbi_pict.db_import*), 149
load_tsv() (*in module thapbi_pict.assess*), 144
lookup_genus() (*in module thapbi_pict.db_import*), 149

lookup_species() (*in module thapbi_pict.db_import*), 149

M

main() (*in module thapbi_pict.assess*), 144
main() (*in module thapbi_pict.classify*), 145
main() (*in module thapbi_pict.conflicts*), 147
main() (*in module thapbi_pict.db_import*), 149
main() (*in module thapbi_pict.denoise*), 147
main() (*in module thapbi_pict.dump*), 152
main() (*in module thapbi_pict.edit_graph*), 152
main() (*in module thapbi_pict.ena_submit*), 162
main() (*in module thapbi_pict.fasta_nr*), 153
main() (*in module thapbi_pict.prepare*), 154
main() (*in module thapbi_pict.sample_tally*), 153
main() (*in module thapbi_pict.summary*), 156
main() (*in module thapbi_pict.taxdump*), 157
make_nr_fasta() (*in module thapbi_pict.prepare*), 154
marker_cut() (*in module thapbi_pict.prepare*), 154
marker_definition (*thapbi_pict.db_orm.SeqSource attribute*), 151
marker_seq (*thapbi_pict.db_orm.SeqSource attribute*), 151
MarkerDef (*class in thapbi_pict.db_orm*), 151
MarkerSeq (*class in thapbi_pict.db_orm*), 151
md5_hexdigest() (*in module thapbi_pict.utils*), 160
md5seq() (*in module thapbi_pict.utils*), 160
merge_paired_reads() (*in module thapbi_pict.prepare*), 154
method_blast() (*in module thapbi_pict.classify*), 145
method_cleanup() (*in module thapbi_pict.classify*), 145
method_dist() (*in module thapbi_pict.classify*), 145
method_identity() (*in module thapbi_pict.classify*), 145
method_substr() (*in module thapbi_pict.classify*), 145
module
    thapbi_pict, 143
    thapbi_pict.assess, 143
    thapbi_pict.classify, 145
    thapbi_pict.conflicts, 147
    thapbi_pict.db_import, 149
    thapbi_pict.db_orm, 151
    thapbi_pict.denoise, 147
    thapbi_pict.dump, 152
    thapbi_pict.edit_graph, 152
    thapbi_pict.ena_submit, 162
    thapbi_pict.fasta_nr, 153
    thapbi_pict.prepare, 154
    thapbi_pict.sample_tally, 153
    thapbi_pict.summary, 156
    thapbi_pict.taxdump, 157
    thapbi_pict.utils, 158
    thapbi_pict.versions, 162

N

none_str() (*in module thapbi_pict.dump*), 152
not_top_species() (*in module thapbi_pict.taxdump*), 157

O

onebp_deletions() (*in module thapbi_pict.utils*), 160
onebp_inserts() (*in module thapbi_pict.utils*), 160
onebp_substitutions() (*in module thapbi_pict.utils*), 160

P

parse_curated_fasta_entry() (*in module thapbi_pict.db_import*), 149
parse_cutadapt_stdout() (*in module thapbi_pict.prepare*), 155
parse_flash_stdout() (*in module thapbi_pict.prepare*), 155
parse_ncbi_fasta_entry() (*in module thapbi_pict.db_import*), 149
parse_ncbi_taxid_entry() (*in module thapbi_pict.db_import*), 150
parse_obitools_fasta_entry() (*in module thapbi_pict.db_import*), 150
parse_sample_tsv() (*in module thapbi_pict.utils*), 160
parse_sintax_fasta_entry() (*in module thapbi_pict.db_import*), 150
parse_species_tsv() (*in module thapbi_pict.utils*), 161
perfect_match_in_db() (*in module thapbi_pict.classify*), 146
perfect_substr_in_db() (*in module thapbi_pict.classify*), 146
prepare_sample() (*in module thapbi_pict.prepare*), 155
primer_clean() (*in module thapbi_pict.utils*), 161

R

read_correction() (*in module thapbi_pict.denoise*), 148
read_summary() (*in module thapbi_pict.summary*), 156
reject_species_name() (*in module thapbi_pict.utils*), 161
run() (*in module thapbi_pict.utils*), 161
run_cutadapt() (*in module thapbi_pict.prepare*), 155
run_flash() (*in module thapbi_pict.prepare*), 155

S

sample_summary() (*in module thapbi_pict.summary*), 156
save_confusion_matrix() (*in module thapbi_pict.assess*), 144
save_mapping() (*in module thapbi_pict.assess*), 144

save_nr_fasta() (*in module thapbi_pict.prepare*), 155
SeqSource (*class in thapbi_pict.db_orm*), 151
setup_blast() (*in module thapbi_pict.classify*), 146
setup_dist2() (*in module thapbi_pict.classify*), 146
setup_dist3() (*in module thapbi_pict.classify*), 146
setup_dist4() (*in module thapbi_pict.classify*), 146
setup_dist5() (*in module thapbi_pict.classify*), 146
setup_dist6() (*in module thapbi_pict.classify*), 146
setup_dist7() (*in module thapbi_pict.classify*), 146
setup_dist8() (*in module thapbi_pict.classify*), 146
setup_dist9() (*in module thapbi_pict.classify*), 146
setup_onebp() (*in module thapbi_pict.classify*), 146
setup_seqs() (*in module thapbi_pict.classify*), 146
source (*thapbi_pict.db_orm.SeqSource attribute*), 151
sp_for_sample() (*in module thapbi_pict.assess*), 144
sp_in_tsv() (*in module thapbi_pict.assess*), 144
species_level() (*in module thapbi_pict.utils*), 161
species_or_species_groups() (*in module thapbi_pict.taxdump*), 157
split_read_name_abundance() (*in module thapbi_pict.utils*), 161
Synonym (*class in thapbi_pict.db_orm*), 152

T

tally_files() (*in module thapbi_pict.assess*), 144
taxid_and_sp_lists() (*in module thapbi_pict.classify*), 147
Taxonomy (*class in thapbi_pict.db_orm*), 152
taxonomy (*thapbi_pict.db_orm.SeqSource attribute*), 152
thapbi_pict
    module, 143
thapbi_pict.assess
    module, 143
thapbi_pict.classify
    module, 145
thapbi_pict.conflicts
    module, 147
thapbi_pict.db_import
    module, 149
thapbi_pict.db_orm
    module, 151
thapbi_pict.denoise
    module, 147
thapbi_pict.dump
    module, 152
thapbi_pict.edit_graph
    module, 152
thapbi_pict.ena_submit
    module, 162
thapbi_pict.fasta_nr
    module, 153
thapbi_pict.prepare
    module, 154
thapbi_pict.sample_tally
    module, 153
thapbi_pict.summary
    module, 156
thapbi_pict.taxdump
    module, 157
thapbi_pict.utils
    module, 158
thapbi_pict.versions
    module, 162

U

unique_or_separated() (*in module thapbi_pict.classify*), 147
unoise() (*in module thapbi_pict.denoise*), 148
usearch() (*in module thapbi_pict.denoise*), 148

V

valid_marker_name() (*in module thapbi_pict.utils*), 161
version_blast() (*in module thapbi_pict.versions*), 162
version_cutadapt() (*in module thapbi_pict.versions*), 163
version_flash() (*in module thapbi_pict.versions*), 163
version_graphviz_fdp() (*in module thapbi_pict.versions*), 163
version_usearch() (*in module thapbi_pict.versions*), 164
version_vsearch() (*in module thapbi_pict.versions*), 164
vsearch() (*in module thapbi_pict.denoise*), 148

W

write_pdf() (*in module thapbi_pict.edit_graph*), 153
write_table() (*in module thapbi_pict.ena_submit*), 162
write_xgmml() (*in module thapbi_pict.edit_graph*), 153