ProPhyle Documentation

Release

Karel Břinda, Kamil Salikhov, Simone Pignotti, Gregory Kucherov

Sep 19, 2017

ProPhyle Manual

CHAPTER 1

Links

GitHub repository - Bug reporting - Contact

CHAPTER 2

Table of contents

Introduction

ProPhyle is a metagenomic classifier based on BWT-index and phylogenetic trees, whose indexing strategy is based on the bottom-up propagation of genomes' k-mers in the tree, assembling contigs at each node and matching using a standard full-text search. The analysis of shared k-mers between NGS reads and the genomes in the index determines which nodes are the best candidates for their classification.

To start using ProPhyle, visit the *Tutorials* or the *Reference* page.

Tutorials

Installation

Prerequisities

ProPhyle is written in Python and C++. It is distributed as a Python package and all C++ auxiliary programs are compiled upon the first execution of the main program. ProPhyle requires the following dependencies:

- Python 3 with ETE3

- GCC 4.8+
- ZLib

Using Conda (recommended)

Environment installation:

```
conda create -c bioconda -n prophyle prophyle
```

Environment activation:

```
source activate prophyle
```

Using PIP

From PyPI:

```
pip install --upgrade prophyle
```

From Git:

```
pip install --upgrade git+https://github.com/karel-brinda/prophyle
```

From PyPI to the current directory:

```
pip install --user prophyle
export PYTHONUSERBASE=`pwd`
export PATH=$PATH:`pwd`/bin
```

Quick test

To quickly test if ProPhyle has been installed correctly, you can create a small index with a small k-mer length:

```
prophyle download bacteria
prophyle index -k 10 -s 0.1 ~/prophyle/bacteria.nw test_idx
```

Default databases

ProPhyle comes with several genome libraries containing RefSeq genomes, augmented with the NCBI taxonomy.

Downloading genomes

These libraries can be downloaded using `prophyle download <library> [<library> ...]`, where `<library>` should be replaced by `bacteria`, `viruses`, or `plasmids`. The command also copies a prebuild Newick/NHX tree for the specified library. If the *-d* parameter is not specified, all files are placed to *~/prophyle*.

To download all viral and bacterial genomes from RefSeq, execute:

```
prophyle download bacteria viruses
```

Index construction

Once a library is downloaded, a ProPhyle index can be constructed using:

```
prophyle index [-g DIR] [-j INT] [-k INT] [-M] [-P] [-K] <tree.nw> [<tree.nw> ...]
↪<index.dir>
```

<tree.nw> is a Newick/NHX for the index. The trees from the previous command are placed in *~/prophyle* and they are called *bacteria.nw*, *viruses.nw*, etc. *<index.dir>* is the directory directory where your index files are going to be placed.

There are multiple other parameters that can be used. *-j* can be used to specify the number of CPU cores used for index construction (all cores are used otherwise). *-k* serves to set the *k*-mer length (31 in default). *-M* activates low complexity regions filtering using DustMasker. Please, ensure that the program is install (try to run *dustmasker*). If multiple trees are used, they are going to be merged. Therefore, a name collision can appear. To prevent such a situation, ProPhyle prepends numerical prefixes to the node names (unless *-P* is used). The *-K* parameter can be used to deactivate *k*-LCP array construction. The resulting index would be slightly smaller, but querying would become much slower.

So the entire command for index construction can look, for instance, like this:

```
prophyle index -k 25 ~/prophyle/bacteria.nw ~/prophyle/viruses.nw my_BV_index
```

Index construction might take several hours, based on the database size, *k* and the number of used cores.

Classification of reads

When the index construction is finished, you can classify your reads using

```
prophyle classify <index_dir> <reads.fq>
```

For the index from above, the command would be:

```
prophyle classify my_BV_index my_reads.fq
```

The assignments are reported in the SAM format. Unless specified otherwise, the *k*-mer length is deduced from the index.

Examples

Quick test (small k, subsampled bacterial database):

```
prophyle download bacteria
prophyle index -k 10 -s 0.1 ~/prophyle/bacteria.nw test_idx
prophyle classify test_idx reads.fq > result.sam
```

Bacterial database (k=31):

```
prophyle download bacteria
prophyle index -k 31 ~/prophyle/bacteria.nw idx_bac
prophyle classify idx_bac reads.fq > result.sam
```

Example for bacterial and viral database (k=31):

```
prophyle download bacteria
prophyle download viruses
prophyle index -k 31 ~/prophyle/bacteria.nw ~/prophyle/viruses.nw idx_bac_vir
prophyle classify idx_bac_vir reads.fq > result.sam
```

Custom NCBI indexes

Download sequences from NCBI

- Create a subdirectory in ProPhyle's main directory, and change the working directory to the newly created one
- Download an `assembly_summary.txt` file from NCBI's FTP server. For RefSeq's bacterial genomes, follow this link. More information about these files can be found here
- Select genomes of interest; the following command will select all complete genomes, but it is sufficient to add more conditions to `awk` to select a subset. Fields 1, 6 and 20 contain respectively the accession number, taxonomic identifier and ftp directory of each genome. `acc2taxid.tsv` will therefore be a tab separated file containing the taxid corresponding to each sequence's accession number. Please refer to NCBI's genomes download FAQ for further information):

```
awk -F "\t" -v OFS="\t" '$12=="Complete Genome" && $11=="latest"\
        {print $1, $6, $20}' assembly_summary.txt >ftpselection.tsv
cut -f 3 ftpselection.tsv | awk 'BEGIN{FS=OFS="/";filesuffix="genomic.fna.gz"}\
        {ftpdir=$0;asm=$10;file=asm"_"filesuffix;print ftpdir,file}' >
↪ftpfilepaths.tsv
cut -f 1,2 ftpselection.tsv >acc2taxid.tsv
```

- Download selected sequences using `parallel --gnu -j 24 -a ftpfilepaths.tsv wget`. Number of jobs (`-j` option) should be changed according to the number of available cores and bandwidth.
- Extract them using `parallel --gnu -j 24 -a ftpfilepaths.tsv gunzip {/}`
- Create a fasta index for each file using `find . -name '*.fna' | parallel --no-notice --verbose samtools faidx {}`

Build a tree

Build a taxonomic tree for the downloaded sequences using:

```
prophyle_homedir/prophyle/prophyle_ncbitree.py <library_name> <library_main_dir>\
             <output_file> acc2taxid.tsv -l <log_file>
```

Taxonomic identifiers are assigned to the sequences first, and then the tree is built using ETE Toolkit and saved with newick format 1. Necessary node attributes are:

- `name`: unique node name (ideally the TaxID of the node)
- `taxid`: unique taxonomic identifier
- `seqname`: names of the sequences sharing the same taxid, separated by @
- `fastapath`: paths of the sequences' fasta files, separated by @ (relative paths from ProPhyle's home directory)
- `infasta_offset`: positions where each sequence starts inside the corresponding fasta files, separated by @
- `base_len`: length of each sequence, separated by @

`seqname`, `infasta_offset` and `base_len` can be found in samtools' fasta index. Other optional attributes are `sci_name`, `named_lineage`, `lineage`, `rank` (more info here). Using ETE library or modifying the `prophyle_ncbitree.py` script it is easy to adapt any tree to match the requirements above.

If taxonomic information is not available, `taxid` attribute can be replaced by the node name. This could be useful to perform classification over custom phylogenetic trees.

Build the index

Run the standard command to build ProPhyle index:

```
prophyle index -k <kmer_length> <tree_1.nw> [<tree_2.nw> ...] <index_dir>
```

Reference

Here you can find format specifications and an automatically generated reference for ProPhyle's CLI.

Formats

Trees

Newick format 1 with NHX annotations, which can be easily created and modified using the ete3 python package.

Classification output

Support for both SAM and Kraken output formats.

Analysis output

- **kraken-report format:**
    1. Percentage of reads covered by the clade rooted at this taxon
    2. Number of reads covered by the clade rooted at this taxon
    3. Number of reads assigned directly to this taxon
    4. A rank code, indicating (U)nclassified, (D)omain, (K)ingdom, (P)hylum, (C)lass, (O)rder, (F)amily, (G)enus, or (S)pecies. All other ranks are simply '-'.
    5. NCBI taxonomy ID
    6. indented scientific name
- **MetaPhlAn2 format:** 1. clades, ranging from taxonomic kingdoms (Bacteria, Archaea, etc.) through species. The taxonomic level of each clade is prefixed to indicate its level: Kingdom: `k__`, Phylum: `p__`, Class: `c__`, Order: `o__`, Family: `f__`, Genus: `g__`, Species: `s__`. Since sequence-based profiling is relative and does not provide absolute cellular abundance measures, clades are hierarchically summed. Each level will sum to 100%; that is, the sum of all kindom-level clades is 100%, the sum of all genus-level clades (including unclassified) is also 100%, and so forth. OTU equivalents can be extracted by using only the species-level `s__` clades from this file (again, making sure to include clades unclassified at this level).
- Custom Centrifuge format:

```
#name                                                          taxID   taxRank
↪ kmerCount   numReads   numUniqueReads   abundance
Wigglesworthia glossinidia endosymbiont of Glossina brevipalpis 36870   leaf
↪ 703004      5981.37    5964             0
```

1. name of a genome, or the name corresponding to a taxonomic ID (the second column) at a rank higher than the strain (e.g., Wigglesworthia glossinidia endosymbiont of Glossina brevipalpis).
2. taxonomic ID (e.g., 36870).
3. taxonomic rank (e.g., leaf).
4. number of k-mers propagated up to the node (e.g., 703004).
5. number of reads classified to this node including multi-classified reads (divided by the number of assignments, e.g., 5981.37).
6. number of reads uniquely classified to this genomic sequence (e.g., 5964).
7. not used yet.

Main program's reference

`prophyle` (list of subcommands)

```
$ prophyle  -h

usage: prophyle.py [-h] [-v]  ...

Program: prophyle (phylogeny-based metagenomic classification)
Version: 0.2.0.0
Authors: Karel Brinda <kbrinda@hsph.harvard.edu>, Kamil Salikhov <kamil.
↪salikhov@univ-mlv.fr>,
                Simone Pignotti <pignottisimone@gmail.com>, Gregory␣
↪Kucherov <gregory.kucherov@univ-mlv.fr>

Usage:   prophyle <command> [options]

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

subcommands:

    download     download a genomic database
    index        build index
    classify     classify reads
    analyze      analyze results
    compress     compress a ProPhyle index (experimental)
    decompress   decompress a compressed ProPhyle index (experimental)
```

`prophyle download`

```
$ prophyle download -h

usage: prophyle.py download [-h] [-d DIR] [-l STR] [-F]
```

```
                          <library> [<library> ...]

positional arguments:
  <library>   genomic library ['bacteria', 'viruses', 'plasmids', 'hmp',
              'all']

optional arguments:
  -h, --help  show this help message and exit
  -d DIR      directory for the tree and the sequences [~/prophyle]
  -l STR      log file
  -F          rewrite library files if they already exist
```

prophyle index

```
$ prophyle index -h

usage: prophyle.py index [-h] [-g DIR] [-j INT] [-k INT] [-l STR] [-s FLOAT]
                         [-F] [-M] [-P] [-K] [-T] [-A]
                         <tree.nw> [<tree.nw> ...] <index.dir>

positional arguments:
  <tree.nw>    phylogenetic tree (in Newick/NHX)
  <index.dir>  index directory (will be created)

optional arguments:
  -h, --help   show this help message and exit
  -g DIR       directory with the library sequences [dir. of the first tree]
  -j INT       number of threads [auto (4)]
  -k INT       k-mer length [31]
  -l STR       log file [<index.dir>/log.txt]
  -s FLOAT     rate of sampling of the tree [no sampling]
  -F           rewrite index files if they already exist
  -M           mask repeats/low complexity regions (using DustMasker)
  -P           do not add prefixes to node names when multiple trees are used
  -K           skip k-LCP construction (then restarted search only)
  -T           keep temporary files from k-mer propagation
  -A           autocomplete tree (names of internal nodes and FASTA paths)
```

prophyle classify

```
$ prophyle classify -h

usage: prophyle.py classify [-h] [-k INT] [-K] [-m {h1,c1}] [-f {kraken,sam}]
                            [-l STR] [-A] [-L] [-P] [-C]
                            <index.dir> <reads1.fq> [<reads2.fq>]

positional arguments:
  <index.dir>      index directory
  <reads1.fq>      first file with reads in FASTA/FASTQ (- for standard␣
↪input)
  <reads2.fq>      second file with reads in FASTA/FASTQ

optional arguments:
  -h, --help       show this help message and exit
```

```
  -k INT            k-mer length [detect automatically from the index]
  -K                use restarted search for matching rather than rolling
                    window (slower, but k-LCP is not needed)
  -m {h1,c1}        measure: h1=hit count, c1=coverage [h1]
  -f {kraken,sam}   output format [sam]
  -l STR            log file
  -A                annotate assignments (using tax. information from NHX)
  -L                use LCA when tie (multiple hits with the same score)
  -P                incorporate sequences and qualities into SAM records
  -C                use C++ impl. of the assignment algorithm (experimental)
```

prophyle analyze

```
$ prophyle analyze -h

usage: prophyle.py analyze [-h] [-s ['w', 'u', 'wl', 'ul']]
                           [-f ['sam', 'bam', 'cram', 'uncompressed_bam',
↪'kraken', 'histo']]
                           {index_dir, tree.nw} <out.pref> <classified.bam>
                           [<classified.bam> ...]

positional arguments:
  {index_dir, tree.nw}     index directory or phylogenetic tree
  <out.pref>               output prefix
  <classified.bam>         classified reads (use '-' for stdin)

optional arguments:
  -h, --help               show this help message and exit
  -s ['w', 'u', 'wl', 'ul']
                           statistics to use for the computation of
                           histograms: w (default) => weighted assignments; u
                           => unique assignments, non-weighted; wl =>␣
↪weighted
                           assignments, propagated to leaves; ul => unique
                           assignments, propagated to leaves.
  -f ['sam', 'bam', 'cram', 'uncompressed_bam', 'kraken', 'histo']
                           Input format of assignments [auto]
```

prophyle compress

```
$ prophyle compress -h

usage: prophyle.py compress [-h] <index.dir> [<archive.tar.gz>]

positional arguments:
  <index.dir>       index directory
  <archive.tar.gz>  output archive [<index.dir>.tar.gz]

optional arguments:
  -h, --help        show this help message and exit
```

prophyle decompress

```
$ prophyle decompress -h

usage: prophyle.py decompress [-h] [-K] archive.tar.gz [output.dir]

positional arguments:
  archive.tar.gz  output archive
  output.dir      output directory [./]

optional arguments:
  -h, --help      show this help message and exit
  -K              skip k-LCP construction
```

Other programs' reference

prophyle_ncbi_tree

```
$ prophyle_ncbi_tree.py -h

usage: prophyle_ncbi_tree.py [-h] [-l log_file] [-r red_factor] [-u root]
                             <library> <library_dir> <output_file> <taxid_
↪map>

Program: prophyle_ncbi_tree Build a taxonomic tree in the New Hampshire␣
↪newick
format #1 for NCBI sequences

positional arguments:
  <library>      directory with the library sequences (e.g. bacteria, viruses
                 etc.)
  <library_dir>  library path (parent of library, e.g. main ProPhyle
                 directory)
  <output_file>  output file
  <taxid_map>    tab separated accession number to taxid map

optional arguments:
  -h, --help     show this help message and exit
  -l log_file    log file [stderr]
  -r red_factor  build reduced tree (one sequence every n)
  -u root        root of the tree (e.g. Bacteria); will exclude sequences
                 which are not its descendants
```

prophyle_assembler

```
$ prophyle_assembler -h


Program:  prophyle_assembler (greedy assembler for ProPhyle)
Contact:  Karel Brinda <karel.brinda@gmail.com>

Usage:    prophyle_assembler [options]

Examples: prophyle_assembler -k 15 -i f1.fa -i f2.fa -x fx.fa
```

```
            - compute intersection of f1 and f2
          prophyle_assembler -k 15 -i f1.fa -i f2.fa -x fx.fa -o g1.fa -o g2.
↪fa
            - compute intersection of f1 and f2, and subtract it from them
          prophyle_assembler -k 15 -i f1.fa -o g1.fa
            - re-assemble f1 to g1

Command-line parameters:
 -k INT   K-mer size.
 -i FILE  Input FASTA file (can be used multiple times).
 -o FILE  Output FASTA file (if used, must be used as many times as -i).
 -x FILE  Compute intersection, subtract it, save it.
 -s FILE  Output file with k-mer statistics.
 -S       Silent mode.

Note that '-' can be used for standard input/output.
```

`prophyle_index` (list of subcommands)

```
$ prophyle_index -h


Program: prophyle_index (alignment of k-mers)
Contact: Kamil Salikhov <kamil.salikhov@univ-mlv.fr>

Usage:   prophyle_index command [options]

Command: build     construct index
         query     query reads against index
```

`prophyle_index build`

```
$ prophyle_index build -h


Usage:   prophyle_index build <prefix>

Options: -k INT    length of k-mer
         -s        construct k-LCP and SA in parallel
         -i        sampling distance for SA
```

`prophyle_index query`

```
$ prophyle_index query -h


Usage:   prophyle_index query [options] <prefix> <in.fq>

Options: -k INT    length of k-mer
         -u        use k-LCP for querying
         -v        output set of chromosomes for every k-mer
         -p        do not check whether k-mer is on border of two contigs, and show␣
↪such k-mers in output
```

```
        -b        print sequences and base qualities
        -l STR    log file name to output statistics
        -t INT    number of threads [1]
```

prophyle_assignment

```
$ prophyle_assignment.py -h

usage: prophyle_assignment.py [-h] [-f {kraken,sam}] [-m {h1,c1}] [-A] [-L]
                              [-X] [-D]
                              <tree.nhx> <k> <assignments.txt>

Implementation of assignment algorithm

positional arguments:
  <tree.nhx>          phylogenetic tree (Newick/NHX)
  <k>                 k-mer length
  <assignments.txt>   assignments in generalized Kraken format

optional arguments:
  -h, --help          show this help message and exit
  -f {kraken,sam}     format of output [sam]
  -m {h1,c1}          measure: h1=hitnumber, c1=coverage [h1]
  -A                  annotate assignments
  -L                  use LCA when tie (multiple hits with the same score)
  -X                  replace k-mer matches by their LCA
  -D                  do not translate blocks from node to tax IDs
```

prophyle_analyze

```
$ prophyle_analyze.py -h

usage: prophyle_analyze.py [-h] [-s ['w', 'u', 'wl', 'ul']]
                           [-f ['sam', 'bam', 'cram', 'uncompressed_bam',
↪'kraken', 'histo']]
                           {index_dir, tree.nw} <out_prefix> <input_fn>
                           [<input_fn> ...]

Program: prophyle_analyze.py

Analyze results of ProPhyle's classification.
Stats:
w: weighted assignments
u: unique assignments (ignore multiple assignments)
wl: weighted assignments, propagated to leaves
ul: unique assignments, propagated to leaves

positional arguments:
  {index_dir, tree.nw}  Index directory or phylogenetic tree
  <out_prefix>          Prefix for output files (the complete file names will
                        be <out_prefix>_rawhits.tsv for the raw hit counts
                        table and <out_prefix>_otu.tsv for the otu table)
  <input_fn>            ProPhyle output files whose format is chosen with the
                        -f option. Use '-' for stdin or multiple files with
```

```
                        the same format (one per sample)

optional arguments:
  -h, --help            show this help message and exit
  -s ['w', 'u', 'wl', 'ul']
                        Statistics to use for the computation of histograms:␣
↪w
                        (default) => weighted assignments; u => unique
                        assignments, non-weighted; wl => weighted␣
↪assignments,
                        propagated to leaves; ul => unique assignments,
                        propagated to leaves.
  -f ['sam', 'bam', 'cram', 'uncompressed_bam', 'kraken', 'histo']
                        Input format of assignments [auto]. If 'histo' is
                        selected the program expects hit count histograms
                        (*_rawhits.tsv) previously computed using prophyle
                        analyze, it merges them and compute OTU table from␣
↪the
                        result (assignment files are not required)
```