phruta: scrapping genbank and assembling phylogenetic trees *

Cristian Román-Palacios *School of Information, University of Arizona, Tucson, Arizona 85721, USA.*
ORCiD: 0000-0003-1696-4886

Current methodological practices for assembling phylogenetic trees often recur to sequence data stored in GenBank. However, understanding molecular and taxonomic availability in GenBank is generally not very straightforward. For instance, the genetic makeup of datasets available in GenBank can strongly differ between genera even within the same family. Similarly, the taxonomic information associated with sequence data in GenBank can be outdated, relative to other databases that mainly focus on the taxonomic side. phruta, a newly developed R package, is designed to improve the user experience and access to information to genetic data stored in GenBank. By using phruta, users are able to (1) quantitatively explore the molecular makeup of particular clades with information in GenBank, (2) assemble curated multi-gene molecular datasets with retrieved and local sequences, and (3) run basic phylogenetic talks, all within R. The structure of the functions implemented in phruta, designed as a workflow, aim to allow users to assemble simple workflows for particular talks, which are in turn expected to increase reproducibility when assembling phylogenies. This paper provides a brief overview on the performance and workflow associated with phruta.

Keywords: R package, Phylogenetics, Reproducibility, Workflow

Background

The phruta R package

The `phruta` package is designed to simplify the basic phylogenetic pipeline in `R`. `phruta` is designed to allow scientists from different backgrounds to assemble their own reproducible phylogenies with as minimal code as possible. All code in `phruta` is run within the same program (`R`) and data from intermediate steps are either stored to the environment or exported locally in independent folders. All code in `phruta` is run within the same environment, an aspect that increases the reproducibility of your analysis. `phruta` looks for potentially (phylogenetically) relevant gene regions for a given set of taxa, retrieves gene sequences, could combine newly downloaded and local gene sequences, performs sequence alignment, phylogenetic inference, and tree dating. `phruta` is largely a wrapper for alternative `R` packages and software.

*Replication files are available on the author's Github account (http://github.com/cromanpa). **Current version**: September 01, 2022; **Corresponding author**: cromanpa94@arizona.edu.

Alternatives to phruta

Similar functionalities for assembling curated molecular datasets for phylogenetic analyses can be found in `phylotaR` and SuperCRUNCH. However, note that `phylotaR` is limited to downloading and curating sequences (e.g. doesn't align sequences). Similarly, `SuperCRUNCH` only curates local sequences. `phruta` is closer to the `SUPERSMART` and its "new" associated `R` workflow `SUPERSMARTR`. However, most of the applications in the different packages that are part of `SUPERSMARTR` are simplified in `phruta`.

phruta in a nutshell

The current release of `phruta` includes a set of eight major functions. All eight functions form a pipeline within `phruta` to output a time-calibrated phylogeny. However, users interested in using their own files at any stage can run each function independently. Note that all the functions for which their primary output are sequences (aligned or unaligned) are listed under `sq.*`. All the files that output phylogenies (time-calibrated or not) are listed under `tree.*`. - First, the distribution of gene sampled for a given organism or set of taxa can be explored using the `acc.gene.sampling` function. This function will return a table that summarizes either the distribution of genes sampled for the search term in general or specifically across species.

- Second, given a list of target organisms, users can retrieve a list of accession numbers that are relevant to their search using `acc.table.retrieve()`. Instead of directly downloading sequences from genbank (see `sq.retrieve.direct()` below), retrieving accession numbers allow users to have more control over the sequences that are being used in the analyses. Note that users can also curate the content of the dataset obtained using `sq.retrieve.direct()`.

- Third, users should download gene sequences. Sequences can be download using the `sq.retrieve.indirec` from the accession numbers retrieved before using the `acc.table.retrieve()` function. This is the preferred option within `phruta`. Additionally, users can directly download gene sequences using the `sq.retrieve.direct()` function. Both `sq.retrieve.indirect()` and `sq.retrieve.direct()` functions save gene sequences in `fasta` files that will be located in a new directory named `0.Sequences`.

- Fourth, `sq.add()` allows users to include local sequences to those retrieved from genbank in the previous step. This function saves all the resulting `fasta` files in two directories, combined sequences in `0.Sequences` and local sequences in `0.AdditionalSequences` (originally downloaded sequences are moved to `0.0.OriginalDownloaded` at this step). Note that `sq.add()` is optional.

- Fifth, the `sq.curate()` function filters out unreliable sequences based on information listed in genbank (e.g. PREDICTED) and on taxonomic information provided by the user. Specifically, this function retrieves taxonomic information from the Global Biodiversity Information Facility (GBIF) database's taxonomic backbone (see alternatives in the advanced vignette to `phruta`). If a given species belongs to a non-target group, this species is dropped from the analyses. This function automatically corrects taxonomy and renames sequences.

- Sixth, `sq.aln()` performs multiple sequence alignment on `fasta` files. Currently, `phruta` uses the `DECIPHER` R package, here. This package allows for adjusting sequence orientation and masking (removing ambiguous sites).

The final two functions in `phruta` focus on tree inference and dating. These two functions depend on external software that needs to be installed (**and tested**) before running. Please make sure both `RAxML` and `PATHd-8` or `treePL` are installed and can be called within `R` using the `system()` function. Note that you can choose between `PATHd-8` and `treePL`. More details on how to install `RAxML` are provided in the phylogenetic vignette of `phruta`. Similarly, we provide details on how to install `PATHd-8` and `treePL` in the same vignette.

- Seventh, the `tree.raxml()` function allows users to perform tree inference under `RAxML` for sequences in a given folder. This is a wrapper to `ips::raxml()` and each of the arguments can be customized. The current release of `phruta` can manage both partitioned and unpartitioned analyses. Starting and constrained trees are allowed.

- Eight, `tree.dating()` enables users to perform time-calibrations of a given phylogeny using `geiger::congruify.phylo()`. `phruta` includes a basic set of comprehensively sampled, time-calibrated phylogenies that are used to extract secondary calibrations for the target phylogeny. Note that sampling in those phylogenies can be examined using `data(SW.phruta)`.

Please make sure you have at least **two** groups in common with each of the phylogenies. Similarly, users can choose to run either `PATHd-8` or `treePL`.

Assembling a molecular dataset for target taxa in phruta

[Need to include exporting data and targetting genes and additional files with sequences]

Let's learn how `phruta` works by assembling a molecular dataset at the species level for a few mammal clades. For this tutorial, assume that we need to build a tree for the following three genera: *Felis, Vulpes*, and *Phoca*. All three genera are classified within the order Carnivora. Both *Felis* and *Vulpes* are classified in different superfamilies within the suborder Fissipedia. *Phoca* is part of another suborder, Pinnipedia. We're going to root our tree with another mammal species, the Chinese Pangolin (*Manis pentadactyla*). Please note that you can select as many target clades and species as you need. However, for simplicity, we will run the analyses using three genera in the ingroup and a single outgroup species.

We have decided the taxonomic make of our analyses. However, we also need to determine the genes that we could use to infer our phylogeny. Fortunately, mammals are extensively studied. For instance, a comprehensive list of potential gene regions to be analyzed for species in this clade is already available in Upham et al (2009). For this tutorial, we will try to find the gene regions are well sampled specifically for the target taxa. I believe that figuring out the best sampled gene regions in genbank, instead of providing names, is potentially more valuable when working with poorly studied groups. For this tutorial, all the objects that are created using functions in `phruta` will be stored in your environment. None of these files will be exported to your working directory. If instead you were interested in exporting the outcomes of particular functions in `phruta`, please follow the tutorial in "To export or not export 'phruta' outputs". Both vignettes aim to assemble a molecular dataset for the same set of taxa under the same approach in `phruta`.

Let's get started with this tutorial by loading `phruta`!

```
library(phruta)
```

Let's now look for the gene regions that are extensively sampled in our target taxa. For this, we will use the `gene.sampling.retrieve()` function in `phruta`. The resulting `data.frame`, named

gs.seqs in this example, will contain the list of full names for genes sampled in genbank for the target taxa.

```
gs.seqs <- gene.sampling.retrieve(organism = c("Felis", "Vulpes", "Phoca", "Manis_pentadactyla")
                                  speciesSampling = TRUE)
```

For the search terms, phruta was able to retrieve the names for r nrow(gs.seqs) gene regions. The frequency estimates per gene are based on inter-specific sampling.

The gene.sampling.retrieve() provides an estimate of the number of species in genbank (matching the taxonomic criteria of the search term) that have sequences for a given gene region. However, this estimate is only as good as the annotations for genes deposited in genbank.

From here, we will generate a preliminary summary of the accession numbers sampled for the combination of target taxa and gene regions. In fact, not all these accession numbers are expected to be in the final (curated) molecular dataset. Using the acc.table.retrieve() function, we will assemble a species-level summary of accession numbers (hence the speciesLevel = TRUE argument). For simplicity, this tutorial will focus on sampling gene regions that are sampled in >30% of the species (targetGenes data.frame).

```
targetGenes <- gs.seqs[gs.seqs$PercentOfSampledSpecies > 30,]
acc.table <- acc.table.retrieve(
            clades  = c('Felis', 'Vulpes', 'Phoca'),
            species = 'Manis_pentadactyla',
            genes = targetGenes$Gene,
            speciesLevel = TRUE
          )
```

The acc.table object is a data.frame that is later on going to be used for downloading locally the relevant gene sequences. In this case, the dataset includes the following information:

Feel free to review this dataset, make changes, add new species, samples, etc. The integrity of this dataset is critical for the next steps so please take your time and review it carefully. Let's just make some minor changes to our dataset:

```
acc.table$Species <- sub("P.", "Phoca ", acc.table$Species, fixed = TRUE)
acc.table$Species <- sub("F.", "Felis ", acc.table$Species, fixed = TRUE)
acc.table$Species <- sub("V.", "Vulpes ", acc.table$Species, fixed = TRUE)
acc.table$Species <- sub("mitochondrial", "", acc.table$Species)
row.names(acc.table) <- NULL
```

Let's check how the new table looks now...

Now, since we're going to retrieve sequences from genbank using an existing preliminary accession numbers table, we will use the `sq.retrieve.indirect()` function in `phruta`. Please note that there are two versions of `sq.retrieve.*` in `phruta` . The one that we're using in this tutorial, `sq.retrieve.indirect()`, retrieves sequences "indirectly" because it necessarily follows the initial step of generating a table summarizing target accession numbers (see the `acc.table.retrieve()` function above). I present the information in this vignette using `sq.retrieve.indirect()` instead of `sq.retrieve.direct()` because the first function is way more flexible. Specifically, it allows for correcting issues *prior* to downloading/retrieving any sequence. For instance, you can add new sequences, species, populations to the resulting data.frame from `acc.table.retrieve()`. Additionally, you could even manually assemble your own dataset of accession numbers to be retrieved using `sq.retrieve.indirect()`. Instead, `sq.retrieve.direct()` does its best to directly (i.e. without input from the user) retrieve sequences for a target set of taxa and set of gene regions. In short, you should be able to catch errors using `sq.retrieve.indirect()` but mistakes will be harder to spot and fix if you're using `sq.retrieve.direct()`.

Now, we still need to retrieve all the sequences from the accessions table generated using `acc.table`. Note that since we have specified `download.sqs = FALSE`, the sequences are returned in a list that is stored in your global environment. If we decide to download the sequences to our working directory using `download.sqs = TRUE`, `phruta` will write all the resulting `fasta` files into a newly created folder `0.Sequences` located in our working directory. The latter option is covered in the "To export or not export 'phruta' outputs" vignette.

```
sqs.downloaded <- sq.retrieve.indirect(acc.table = acc.table, download.sqs = FALSE)
```

We're now going to make sure that we include only sequences that are reliable and from

species that we are actually interested in analyzing. We're going to use the `sq.curate()` function for this. We will provide a list of taxonomic criteria to filter out incorrect sequences (`filterTaxonomicCriteria` argument). For instance, we could simply provide a vector of the genera that we're interested in analyzing. Note that the outgroup's name should also be included among the target taxa. If the taxonomic information for a sequence retrieved from genbank does not match with any of these strings, this species will be dropped. You will have to specify whether sampling is for animals or plants (`kingdom` argument). Finally, you might have already noticed that, sometimes, gene regions have alternative names. In our case, we are going to merge "cytochrome oxidase subunit 1" and "cytochrome c oxidase subunit I" into a single file named `COI`. To merge gene files during the curation step, you will have to provide a named list to the `mergeGeneFiles` argument of the `sq.curate()` function. This named list (`tb.merged` in our tutorial) will have a length that equals the number of final files that will result from merging the target gene files. Note that, since we're not downloading anything to our working directory, we need to pass our downloaded sequences (`sqs.downloaded` object generated above using the `sq.retrieve.indirect()` function) to the `sqs.object` argument in `sq.curate()`.

```
tb.merged <- list('COI' = c("cytochrome oxidase subunit 1", "cytochrome c oxidase subunit I"))
sqs.curated <- sq.curate(filterTaxonomicCriteria = 'Felis|Vulpes|Phoca|Manis',
                         mergeGeneFiles = tb.merged,
                         kingdom = 'animals',
                         sqs.object = sqs.downloaded,
                         removeOutliers = FALSE)
```

Running the `sq.curate()` function will create an object of class list that includes (1) the curated sequences with original names, (2) the curated sequences with species-level names (`renamed_*` prefix), (3) the accession numbers table (`AccessionTable`), and (4) a summary of taxonomic information for all the species sampled in the files (`Taxonomy.csv`).

Now, we'll align the sequences that we just curated. For this, we just use `sq.aln()` with default parameters. We're again passing the output from `sq.curate()`, `sqs.curated`, using the `sqs.object` argument in `sq.aln()`.

```
sqs.aln <- sq.aln(sqs.object = sqs.curated)
```

The resulting multiple sequence alignments will be saved to `sqs.aln()` object, a list. For each of the gene regions, we will have access to the original alignment (`Aln.Original`), the masked one (`Aln.Masked`), and information on the masking process.

Note that we could use these resulting alignments to infer phylogenies. We cover these steps within `phruta` in another vignette: "Phylogenetics with the `phruta` R package". For now, let's wrap up and plot one of our cool alignments. Let's first check the raw alignments.

Now, the masked alignments...

Basic phylogenetics with prhuta

Phylogenetic inference with `phruta` and `RAxML`

Phylogenetic inference is conducted using the `tree.raxml()` function. We need to indicate where the aligned sequences are located (`folder` argument), the patterns of the files in the same folder (`FilePatterns` argument; "`Masked_`" in our case). We'll run a total of 100 boostrap replicates and set the outgroup to "Manis_pentadactyla".

```
tree.raxml(folder='2.Alignments',
           FilePatterns= 'Masked_',
           raxml_exec='raxmlHPC',
           Bootstrap=100,
           outgroup ="Manis_pentadactyla")
```

The trees are saved in `3.Phylogeny`. Likely, the bipartitions tree, "RAxML_bipartitions.phruta", is the most relevant. `3.Phylogeny` also includes additional `RAxML`-related input and output files.

Finally, let's perform tree dating in our phylogeny using secondary calibrations extracted from Scholl and Wiens (2016). This study curated potentially the most comprenhensive and reliable set of trees to summarize the temporal dimension in evolution across the tree of life. In `phruta`, the trees from Scholl and Wiens (2016) were renamed to match taxonomic groups.

Tree dating in phruta

Tree dating is performed using the `tree.dating()` function in `phruta`. We have to provide the name of the folder containing the `1.Taxonomy.csv` file created in `sq.curate()`. We also have to indicate the name of the folder containing the `RAxML_bipartitions.phruta` file. We will scale our phylogeny using `treePL`.

```
tree.dating(taxonomyFolder="1.CuratedSequences",
            phylogenyFolder="3.Phylogeny",
            scale='treePL')
```

Running this line will result in a new folder `4.Timetree`, including the different time-calibrated phylogenies obained (if any) and associated secondary calibrations used in the analyses. We found only a few overlapping calibration points (family-level constraints):

Here's the resulting time-calibrated phylogeny. The whole process took ~20 minutes to complete on my computer (16 gb RAM, i5).

Advanced methods with phruta As explained in the brief intro to `phruta`, the `sq.curate()` function is primarily designed to curate taxonomic datasets using `gbif`. Alto `gbif` is extremely fast and efficient, it is largely designed to deal with animals and plants. If you're interested in using the `gbif` backbone for curating sequence regardless of the kingdom use the following approach:

```
taxonomy.retrieve(species_names=c("Felis_catus", "PREDICTED:_Vulpes",
                  "Phoca_largha", "PREDICTED:_Phoca" ,
                  "PREDICTED:_Manis" , "Felis_silvestris" , "Felis_nigripes"),
                  database='gbif', kingdom=NULL)
```

Note that the `kingdom` argument is set to `NULL`. However, as indicated in the first vignette, `gbif` is efficient for retrieving accurate taxonomy when we provide details on the `kingdom`. Given that all the species we're interested in are animals, we could just use:

```
taxonomy.retrieve(species_names=c("Felis_catus", "PREDICTED:_Vulpes",
                  "Phoca_largha", "PREDICTED:_Phoca" ,
                  "PREDICTED:_Manis" , "Felis_silvestris" , "Felis_nigripes"),
                  database='gbif', kingdom='animals')
```

We could also do the same for plants by using `plants` instead of `animals` in the `kingdom` argument. Now, what if we were interested in following other databases to retrieve taxonomic information for the species in our database? The latest version of `phruta` allow users to select the desired database. The databases follow the `taxize::classification()` function. Options are: `ncbi`, `itis`, `eol`, `tropicos`, `nbn`, `worms`, `natserv`, `bold`, `wiki`, and `pow`. Please select only one. Note that the `gbif` option in `taxize::classification()` is replaced by the internal `gbif` in `phruta`.

Now, let's assume that we were interested in curating our database using `itis`:

```
taxonomy.retrieve(species_names=c("Felis_catus", "PREDICTED:_Vulpes",
                  "Phoca_largha", "PREDICTED:_Phoca" ,
                  "PREDICTED:_Manis" , "Felis_silvestris" , "Felis_nigripes"),
                  database='itis')
```

Using alternative databases is sometimes desirable. Please make sure you review which the best database is for your target group is before selecting one.

Creating taxonomic constraints in `phruta`

For different reasons, phylogenetic analyses sometimes require of tree constraints. `phruta` can automatically generate trees in accordance to taxonomy and a backbone topology. We divide constraint trees into two: (1) ingroup+outgroup and (2) particular clades.

ingroup + outgroup

In this constraint type, `phruta` will create monophyletic groups for each of the taxonomic groups in the database (for selected target columns). Finally, it will generate tree with the same topology provided in the `Topology` argument. The user will provide the species names of the outgroup taxa as a vector of string that should fully match the names in the taxonomy file.

```
tree.constraint(
                taxonomy_folder = "1.CuratedSequences",
                targetColumns = c("kingdom", "phylum", "class", "order",
                                  "family", "genus", "species_names"),
                Topology = "((ingroup), outgroup);",
                outgroup = "Manis_pentadactyla"
 )
```

Particular clades

In this constraint type, `phruta` will create a constraint tree for particular clades. For instance, let's assume that we only need to create a tree constraining the monophyly within two genera and their sister relationships:

```
tree.constraint( taxonomy_folder = "1.CuratedSequences",
                 targetColumns = c("kingdom", "phylum", "class",
                                   "order", "family", "genus", "species_names"),
                 Topology = "((Felis), (Phoca));"
 )
```

Note that the key aspect here is the `Topology` argument. It is a newick tree.

Running PartitionFinder in `phruta`

With the current version of `phruta`, users are able to run `PartitionFinder` v1 within `R`. For this, users should provide the name of the folder where the alignments are stored, a particular pattern in the file names (`masked` in our case), and which models will be run in `PartitionFinder`. This function will download `PartitionFinder`, generate the input files, and run it all within R. The output files will be in a new folder within the working directory.

```
sq.partitionfinderv1(folderAlignments = "2.Alignments",
                     FilePatterns = "Masked",
```

```
                    models = "all"
 )
```

Unfortunately, the output files are not integrated with the current `phruta` pipeline. This will be part of a new release. However, users can still perform gene-based partitioned analyses within `RAxML` or can use PartitionFinder's output files to inform their own analyses outside `phruta`.

Partitioned analyses in RAxML

Users can now run partitioned analyses in `RAxML` within `phruta`. This approach is implemented by setting the `partitioned` argument in `tree.raxml` to `TRUE`. For now, partitions are based on the genes are being analyzed. The same model is used to analyze each partition. More details on partitioned analyses can be customized by passing arguments in `ips::raxml`.

```
tree.raxml(folder = "2.Alignments", FilePatterns = "Masked",
           raxml_exec = "raxmlHPC", Bootstrap = 100,
           outgroup = "Manis_pentadactyla",
           partitioned=T
)
```

Identifying rogue taxa

`phruta` can help users run `RogueNaRok` implemented in the `Rogue` R package. Users can then examine whether rogue taxa should be excluded from the analyses. `tree.roguetaxa()` uses the bootstrap trees generated using the `tree.raxml()` function along with the associated best tree to identify rogue taxa.

```
tree.roguetaxa(folder = "3.Phylogeny")
```

Reproducibility with phruta Users can choose to share the script they used to run the analyses (e.g. assemble their molecular dataset) and the associated workspace.

Performance