treeoclock

Release 0.0.1

Lars Berling, Lena Collienne, Jordan Kettles

Jun 22, 2021

CONTENTS:

CHAPTER

ONE

WORKING WITH TIME TREES

Contents

1.1 The TimeTree class

A `TimeTree` can be initialized with a given newick string using the ete3.Tree constructor and its format options. Additionally, a `TREE` object (*Classes for the c library*) is generated and saved in the `TimeTree` and used for efficient distance computations.

1.1.1 TimeTree attributes

Method	Description
`TimeTree.etree`	returns the `ete3.Tree` object
`TimeTree.ctree`	returns the respective `TREE` object
`len(TimeTree)`	returns the number of leaves of the `TimeTree`
`TimeTree.fp_distance(t)`	returns the findpath distance to another `TimeTree` *t*
`TimeTree.fp_path(t)`	returns a *list* of `TREE` objects
`TimeTree.get_newick(format)`	returns the *write()* function of the `ete3.Tree` with the specified *format*, defaults to *format=5*
`TimeTree.copy()`	returns a deep copy of the current `TimeTree` (specifically used to generate a new `TREE` object)
`TimeTree.neighbours()`	returns a list of `TimeTree`'s containing all neighbours at distance *1*
`TimeTree.rank_neighbours()`	returns a list of `TimeTree`'s containing only neighbours one *rank move* away
`TimeTree.nni_neighbours()`	returns a list of `TimeTree`'s containing only neighbours one *NNI move* away

This is an example of how to access the different attributes of a TimeTree object:

```
from treeoclock.trees.time_trees import TimeTree


# Initialize a time tree from a newick string
tt = TimeTree("((1:3,5:3):1,(4:2,(3:1,2:1):1):2);")

tt.ctree  # the TREE class object

tt.etree  # the ete3.Tree object

len(tt)  # Number of leaves in the tree tt --> 5

tt.fp_distance(tt)  # Distance to another TimeTree --> 0

tt.fp_path(tt)  # A shortest path to another TimeTree --> []

tt.get_newick()  # Returns the newick string in ete3 format=5

ttc = tt.copy()  # ttc contains a deep copy of the TimeTree tt

tt.neighbours()  # a list of TimeTree objects each at distance one to tt

tt.rank_neighbours()  # list of TimeTree obtained by doing all possible rank moves on␣
↪tt

tt.nni_neighbours()  # list of TimeTree obtained by doing all possible NNI moves on tt
```

1.1.2 ete3 functionalities

Via the `ete3.Tree` object the respective function of the `ete3` package are available for a `TimeTree` object. For example drawing and saving a tree to a file:

```
from treeoclock.trees.time_trees import TimeTree

tt = TimeTree("((1:3,5:3):1,(4:2,(3:1,2:1):1):2);")

# Automatically save the tree to a specific file_path location
tt.etree.render('file_path_string')

# Defining a layout to display internal node names in the plot
def my_layout(node):
    if node.is_leaf():
        # If terminal node, draws its name
        name_face = ete3.AttrFace("name")
    else:
        # If internal node, draws label with smaller font size
        name_face = ete3.AttrFace("name", fsize=10)
    # Adds the name face to the image at the preferred position
    ete3.faces.add_face_to_node(name_face, node, column=0, position="branch-right")

ts = ete3.TreeStyle()
ts.show_leaf_name = False
ts.layout_fn = my_layout
ts.show_branch_length = True
ts.show_scale = False

# Will open a separate plot window, which also allows interactive changes and saving␣
↪the image
tt.etree.show(tree_style=ts)
```

See the `ete3` documentation for more options.

1.2 The TimeTreeSet class

A `TimeTreeSet` is an iterable list of `TimeTree` objects, which can be initialized with a nexus file.

Method	Description
`TimeTreeSet.map`	a dictionary conataining the taxa to integer translation from the nexus file
`TimeTreeSet.trees`	a list of `TimeTree` objects
`TimeTreeSet[i]`	returns the `TimeTree` at `TimeTreeSet.trees[i]`
`len(TimeTreeSet)`	returns the number of trees in the list `TimeTreeSet.trees`
`TimeTreeSet.fp_distance(i, j)`	returns the distances between the trees at postition i and j
`TimeTreeSet.fp_path(i, j)`	returns a shortest path (list of `TREE`) between the trees at postition i and j

1.2.1 Reading Trees

A TimeTreeSet object can be initialized with a path to a nexus file.

```
from treeoclock.trees.time_trees import TimeTreeSet


# Initializing with a path to a nexus tree file
tts = TimeTreeSet("path_to_nexus_file.nex")

tts.map  # a dictionary keys:int and values:string(taxa)

tts.trees  # A list of TimeTree objects

for tree in tts:
    # tree is a TimeTree object
    ...
tts[0]  # trees are accessible via the index

len(tts)  # Returns the number of trees in the TimeTreeSet object

tts.fp_distance(i, j)  # Returns the distance between trees i and j
tts.fp_path(i, j)  # Returns a shortest path between trees i and j
```

1.2.2 Writing trees

Still WIP

1.3 Random trees

STILL WIP

1.4 Combining multiple TimeTreeSets

Still WIP

1.5 General Functions

A list of the direct functions and their arguments.

Function	Description
`time_trees.neighbourhood(tree)`	returns a list of `TimeTree` objects containing the one-neighbours of tree
`time_trees.get_rank_neighbours(tree)`	returns a list of `TimeTree` objects containing the rank neighbours of tree
`time_trees.get_nni_neighbours(tree)`	returns a list of `TimeTree` objects containing the NNI neighbours of tree
`time_trees.read_nexus(file)`	returns a list of `TimeTree` objects contained in given the nexus file
`time_trees.get_mapping_dict(file)`	returns a dictionary containg the taxa to integer transaltion of the given file
`time_trees.findpath_distance(t1, t2)`	Computes the distance between t1 and t2
`time_trees.findpath_path(t1, t2)`	Computes the path between t1 and t2

Note: Both functions `time_trees.findpath_distance(t1, t2)` and `time_trees.findpath_path(t1, t2)` can be called with t1 and t2 being either a `TREE`, `TimeTree` or `ete3.Tree`

1.6 Classes for the c library

These classes are found in the `_ctrees.py` module. The corresponding CDLL c library is generated from `findpath.c`.

1.6.1 NODE

- `parent`: index of the parent node (int, defaults to -1)
- `children[2]`: index of the two children ([int], defaults to [-1, -1])
- `time`: Time of the node (int, defaults to 0)

Note: The attribute `time` is currently not being used!

1.6.2 TREE

- `num_leaves`: Number of leaves in the tree (int)
- `tree`: Points to a `NODE` object (POINTER(`NODE`))
- `root_time`: Time of the root `Node` (int)

Note: The attribute `root_time` is currently not being used!

1.6.3 TREELIST

- `num_trees`: Number of trees in the list (int)
- `trees`: List of trees (POINTER(`TREE`))

1.7 Class converter functions

These are found in `_converter.py`.

Function	Description
`_converter.ete3_to_ctree(tree)`	traverses an `ete3.Tree` and construct the correct `TREE`
`_converter.ctree_to_ete3(ctree)`	recursively traverses a `TREE` and generates an `ete3.Tree`

CHAPTER

TWO

SUMMARIZING TREES

Contents

2.1 The Centroid class

TODO: Change the start default to the Frechet Mean algorithm instead of last

```
class treeoclock.summary.Centroid(variation="greedy", n_cores=None,
                                  select='random', start='last')
```

Applying a variation of the centroid algorithm on a set of trees.

2.1.1 Centroid attributes

Method	Description
`Centroid.variation`	Accesses the variation parameter which has to be in the list of specified variations.
`Centroid.n_cores`	Accesses the number of Threads that will be used to compute the SoS value, using ThreadPool.
`Centroid.select`	Accesses the selection parameter, in case of multiple options choose a selected tree i.e. the first, last or a random tree.
`Centroid.start`	Accesses the start parameter, specifying at which point to start the centroid computation.
`Centroid.compute_centroid(trees)`	Computes the centroid for a given set of trees, using the options that were selected via the other attributes of the `Centroid`.

Examples of how to set and use the attributes of a `Centroid`:

```
from treeoclock.trees.time_trees import TimeTreeSet
from treeoclock.summary.centroid import Centroid


# Initializing an empty Centroid class
mycen = Centroid()

mycen.variation = "inc_sub"  # Changing the variation

mycen.n_cores = 4  # Setting the number of Threads to use

mycen.select = "first"  # Selection parameter set to the first tree found

mycen.start = 6  # Starting the algorithm from the sixth tree in the set (Error if␣
↪non-existant)

# Computing the centroid for an empty TimeTreeSet
cen_tree, sos = mycen.compute_centroid(TimeTreeSet())
```

2.1.2 Centroid variations

The variation parameter of a `Centroid` has to be one in ["inc_sub", "greedy"] (TODO: Still WIP).

Variation	Description
greedy	Computes a centroid via the greedy path and neighbourhood search. Only considering the tree with the most imporved SoS value in each iteration.
inc_sub	Starts with a subsample of trees from the set, computes the greedy centroid variant and adds more trees to the subsample until all trees are part of the sample.
WIP	WIP
WIP	WIP
WIP	WIP

2.1.3 Computing the SoS

2.1.4 Sampling the neighbourhood

2.2 Frechet Mean

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search