
Pylogeny Documentation

Release 0.3.4.6

Alex Safatli

March 15, 2015

CONTENTS

1	pylogeny package	3
1.1	Submodules	3
1.2	Module contents	25
2	Indices and tables	27
	Python Module Index	29
	Index	31

Contents:

PYLOGENY PACKAGE

1.1 Submodules

1.1.1 pylogeny.JSONWriter module

Serialize a phylogenetic landscape into a JSON object.

```
class pylogeny.JSONWriter.JSONWriter(ls, name)
    Bases: pylogeny.landscapeWriter.landscapeWriter
    __init__(ls, name)
    getCompleteLandscape()
    getJSON()
    getOnlyImprovements(groups=None)
    nodeToJSON(node)
```

1.1.2 pylogeny.alignment module

Handle input biological sequence alignment files for the purposes of phylogenetic inference. Will read all types of alignment files by utilizing the P4 python phylogenetic library.

```
class pylogeny.alignment.alignment(inal=None)
    Bases: object

    Wrap a biological sequence alignment to enable functionality necessary for phylogenetic inference. Makes use
    of temporary files; requires to be closed once no longer needed.

    __init__(inal=None)
        Instantiate an object intended to wrap an alignment for the purposes of running phylogenetic inference.

        Parameters inal – An alignment file path (most formats are accepted).

    bootstrap()
        Perform bootstrapping on the alignment data.

    close()
        Delete all temporary files and clear data.

    getAlignment()
        Acquire the alignment data structure (P4 module).
```

getApproxMLNewick ()

Get a tree in newick format via use of FastTree that serves as an approximation of the maximum likelihood tree for this data.

getApproxMLTree ()

Get a tree object for an approximation of the maximum likelihood tree for this data using FastTree.

getDim ()

Return the dimensionality of the sequence alignment (how many different types of characters).

getFASTA ()

Get (and create if not already) a path to a temporary FASTA file. This will be deleted upon closure of the alignment instance.

getFastTreeNewick ()

Alias for the “getApproxMLNewick()” function.

getNumSeqs ()

Return the number of sequences that are present in the sequence alignment.

getSequence (*i*)

Acquire the *i*th sequence.

getSize ()

Return the size of the alignment, or how many characters there are in each respective item in the alignment.

getStateModel ()

getTaxa ()

Return taxa names.

toStrList ()

Get all sequences as a list of strings.

class `pylogeny.alignment.phylipFriendlyAlignment` (*inal=None*)

Bases: `pylogeny.alignment.alignment`

An alignment object that renames all comprising taxa in order to be able to be written as a strict Phylip file.

__init__ (*inal=None*)

getPhylip ()

Get a path to a temporary Phylip file. This will be deleted upon closure of the alignment instance.

getProperName (*n*)

Return the actual name for an integer-based sequence name that was reassigned at initialization.

getTaxa ()

Return current taxa names in the alignment.

reassignFromReinterpretedNewick (*tr*)

Replace all proper names with reassigned names in Newick tree.

recreateObject ()

Reintializes the object.

reinterpretNewick (*tr*)

Replaces all reassigned names to proper names in Newick tree.

writeProperNexus (*wri*)

Write a Nexus file with proper names.

1.1.3 pylogeny.base module

Definitions for generalized containers and objects used by other structures in this framework.

`pylogeny.base.longest_common_substring(s1, s2)`

Simplified, traditional LCS algorithm implementation.

class `pylogeny.base.patriciaTree`

Bases: `pylogeny.base.trie`

Defines a PATRICIA tree (condensed trie) across a range of strings.

delete (*seq*)

Remove a sequence from the PATRICIA tree. Will not remove added characters to alphabet.

insert (*seq*)

Dynamically insert a sequence into the PATRICIA tree. Returns the unique index in the tree for that string.

search (*seq*)

Search for a sequence in the PATRICIA tree. Returns its position in addition sequence if it exists. Else, returns 0.

class `pylogeny.base.treeBranch` (*parent=None, child=None, label=''*)

Bases: `object`

A branch in a tree.

__init__ (*parent=None, child=None, label=''*)

child = `None`

getChild ()

getLabel ()

getParent ()

label = `'`

parent = `None`

class `pylogeny.base.treeNode` (*lbl=None, children=None, parent=None*)

Bases: `object`

A node in a tree.

__init__ (*lbl=None, children=None, parent=None*)

addChild (*item*)

children = `None`

getChildByIndex (*i*)

getChildren ()

getLabel ()

getParent ()

isInternalNode ()

isLeaf ()

label = `None`

parent = `None`

```
class pylogeny.base.treeStructure (root=None)
```

```
    Bases: _abcoll.Container
```

Defines a base collection of treeNodes and treeBranches in a hierarchical tree structure.

```
    __init__ (root=None)
```

```
    getAllLeaves ()
```

```
    getAllNodes ()
```

```
    getPostOrderTraversal ()
```

```
    getRoot ()
```

Return the top-level, root, node of the tree.

```
    static leaves (root)
```

```
    static nodes (root)
```

```
    static postOrderTraversal (root)
```

```
    root = None
```

```
class pylogeny.base.trie
```

```
    Bases: _abcoll.Sized, pylogeny.base.treeStructure
```

Defines a trie across a range of strings.

```
    __init__ ()
```

```
    alphabet = None
```

```
    count = 0
```

```
    delete (seq)
```

Remove a sequence from the trie. Will not remove added characters to alphabet.

```
    getAlphabet ()
```

```
    getRoot ()
```

```
    insert (seq)
```

Dynamically insert a sequence into the trie.

```
    nextLabel = 1
```

```
    root = None
```

```
    search (seq)
```

Search for a sequence in the trie. Returns true if it exists.

```
class pylogeny.base.trieNode (lbl=None, children=None, parent=None)
```

```
    Bases: pylogeny.base.treeNode
```

A subclass of treeNode that allows for checking non-zero members amongst children branches and other conveniences.

```
    getNonEmptyChildrenBranchLabels ()
```

```
    getNonEmptyChildrenBranches ()
```

Acquire a list of all non-empty children.

```
    getNonEmptyChildrenNodes ()
```

Acquire a list of all non-empty children.

```
    getParentNode ()
```

iterNonEmptyChildrenNodes ()

Iterate over all children that are not empty.

numEmptyChildrenNodes ()

Acquire the number of children nodes that are marked 0 or nonexistent.

setChildNode (*child*, *newchild*)

1.1.4 pylogeny.database module

Connect, access, + manipulate external tree data from a remote SQL server or from a sqlite file.

class `pylogeny.database.DatabaseLandscape` (*ali*, *starting_tree=None*, *root=True*, *operator='SPR'*)

Bases: `pylogeny.landscape.landscape`

Abstract the landscape to one comprising a landscape.

getNode (*i*)

class `pylogeny.database.SQLDatabase` (*host*, *user*, *pw*, *db*)

Bases: `pylogeny.database.database`

Database object to allow reading from a MySQL database.

__init__ (*host*, *user*, *pw*, *db*)

close ()

connect ()

getColumns (*table*)

Return column information for a given table.

getTables ()

query (*q*)

querymany (*q*, *i*)

class `pylogeny.database.SQLExhaustiveLandscape` (*dbobj*, *aliname*)

Bases: `pylogeny.database.DatabaseLandscape`

__init__ (*dbobj*, *aliname*)

exploreRandomTree (*i*)

exploreTree (*i*)

getDatabaseNode (*i*)

class `pylogeny.database.SQLiteDatabase` (*filepath*)

Bases: `pylogeny.database.database`

__init__ (*filepath*)

close ()

getColumns (*table*)

Return column information for a given table.

getTables ()

query (*q*)

querymany (*q*, *i*)

```
class pylogeny.database.SQLiteLandscape (dbobj)
    Bases: pylogeny.landscape.landscape

    Allow random access of all landscape data from an sqlite file found on the hard disk.

    __init__ (dbobj)

class pylogeny.database.database
    Bases: object

    Allow interfacing with a SQL/sqlite database.

    close ()

    cursor = None

    filterRecords (table, condn)
        Get all records from a given table following a condition.

    getColumns (table)

    getHeaders (table)
        Get only header names for a given table's columns.

    getRecords (table)
        Get all records from a given table in the database.

    getRecordsAsDict (table)
        Acquires records using getRecords() and then leverages access using a dictionary data structure.

    getRecordsColumn (table, col)
        Get all data for a single column from records for a table.

    getTables ()

    insertRecord (tablename, record)

    insertRecords (tablename, items)

    isEmpty ()
        Determine if the database is empty.

    iterRecords (table)
        Get a record, one at a time, from a table in the database.

    newTable (tablename, *args)

    query (q)

    querymany (q, i)
```

1.1.5 pylogeny.executable module

Defines an interface to manage interfacing with the system for respective application calls and implements some of these for executables such as FastTree and RAxML. Currently requires a UNIX-like environment (e.g., Mac OS X or a Linux-based environment).

```
class pylogeny.executable.aTemporaryDirectory (dir=None)
    Bases: object

    A class intended to be used as a context manager that allows Python to run in a temporary directory for a finite period of time.

    __init__ (dir=None)
```

class `pylogeny.executable.consel` (*treeSet, alignment, name*)

Bases: `pylogeny.executable.executable`

Denotes a single run of the CONSEL workflow in order to acquire a confidence interval and perform an AU test on a set of trees. Requires CONSEL to be installed.

__init__ (*treeSet, alignment, name*)

getInstructionString ()

getInterval ()

Compute the AU test. Return the interval of trees.

`pylogeny.executable.exeExists` (*cmd*)

Determines whether a function exists in a UNIX environment.

class `pylogeny.executable.executable`

Bases: `object`

An abstract class for the instantiation and running of a single instance for a given application.

exeName = `None`

getInstructionString ()

run ()

Perform a run of this application.

class `pylogeny.executable.fasttree` (*inp_align, out_file=None, isProtein=True*)

Bases: `pylogeny.executable.executable`

Denotes a single run of the FastTree executable in order to acquire an approximate maximum likelihood tree for the input alignment. See <http://www.microbesonline.org/fasttree/> for more information on FastTree. Requires FastTree to be installed.

__init__ (*inp_align, out_file=None, isProtein=True*)

exeName = `'fasttree'`

getInstructionString ()

class `pylogeny.executable.raxml` (*inp_align, out_file, model=None, is_Protein=True, interTrees=False, alg=None, startingTree=None, rapid=False, slow=False, optimizeBootstrap=False, numboot=100, log=None, wdir=None*)

Bases: `pylogeny.executable.executable`

Denotes a single run of the RAxML executable. See <http://sco.h-its.org/exelixis/software.html> for more information on RAxML. Requires RAxML to be installed.

__init__ (*inp_align, out_file, model=None, is_Protein=True, interTrees=False, alg=None, startingTree=None, rapid=False, slow=False, optimizeBootstrap=False, numboot=100, log=None, wdir=None*)

exeName = `'raxmlHPC'`

getInstructionString ()

runFunction (*alg*)

class `pylogeny.executable.rspr` (*treeA, treeB, algorithm='', overlap=True*)

Bases: `pylogeny.executable.executable`

Denotes a single run of the rSPR executable by Dr. Chris Whidden (2014), a software package for computing rooted subtree-prune-and-regraft (SPR) distances. See <http://kiwi.cs.dal.ca/Software/RSPR>. Requires the executable to be on PATH.

```
RSPR_ALG_APPROX = '-approx'
RSPR_ALG_BB = '-bb'
RSPR_ALG_DEFAULT = ''
RSPR_ALG_FPT = '-fpt'

__init__(treeA, treeB, algorithm='', overlap=True)
    Algorithm choices are defined in this class. If overlap is set to True, will attempt to consolidate taxa names
    such that they are overlapping (otherwise, RSPR will return an error if they do not match).

exeName = 'rspr'

getInstructionString()

getSPRDistance()

class pylogeny.executable.treepuzzle(ali, treefile)
    Bases: pylogeny.executable.executable

    Wrap TREE-PUZZLE in order to create an intermediate file for CONSEL to read and assign confidence to a set
    of trees. Requires TREE-PUZZLE to be installed.

    __init__(ali, treefile)

    exeName = 'puzzle'

    getInstructionString()

    getSiteLikelihoodFile()
```

1.1.6 pylogeny.heuristic module

Define the interface for a heuristic in order to implement any manner of heuristic for a combinatorial problem that can be abstracted into a state graph. In this case, a phylogenetic tree space.

```
class pylogeny.heuristic.RAxMLIdentify(ls, startNode, workdir='.xml')
    Bases: pylogeny.heuristic.phylogeneticLinearHeuristic

    RAxML-driven landscape evaluation of intermediate checkpoint trees output from the RAxML executable.

    __init__(ls, startNode, workdir='.xml')

    explore()

class pylogeny.heuristic.heuristic(G=None, start=None)
    Bases: object

    A base interface for a heuristic that explores a state graph.

    __init__(G=None, start=None)

    explore()

    getStartState()

    getStateGraph()

class pylogeny.heuristic.likelihoodGreedy(ls, startNode)
    Bases: pylogeny.heuristic.phylogeneticLinearHeuristic

    Greedy (hill-climbing) landscape exploration by comparison of likelihood.

    __init__(ls, startNode)
```

```

explore ()
    Perform greedy search of the landscape using a method of greed via likelihood.

class pylogeny.heuristic.parsimonyGreedy (ls, startNode)
    Bases: pylogeny.heuristic.phylogeneticLinearHeuristic

    Greedy (hill-climbing) landscape exploration by comparison of parsimony.

    __init__ (ls, startNode)

    explore ()
        Perform greedy search of the landscape using a method of greed via parsimonious criterion.

class pylogeny.heuristic.phylogeneticLinearHeuristic (ls, startNode)
    Bases: pylogeny.heuristic.heuristic

    A base class for a heuristic that works on a phylogenetic landscape and only possesses a single path (of search).

    __init__ (ls, startNode)

    bestTree = None

    getBestTree ()

    getPath ()

    path = []

class pylogeny.heuristic.smoothGreedy (ls, startNode)
    Bases: pylogeny.heuristic.phylogeneticLinearHeuristic

    Parsimony-driven greedy landscape exploration by comparison of likelihoods.

    __init__ (ls, startNode)

    explore ()
        Perform greedy search of the landscape using a method of greed via parsimonious criterion and then
        performing final smoothing via likelihood on top 10% of 1-SPR neighbors ranked on basis of parsimony.

```

1.1.7 pylogeny.landscape module

Encapsulate a phylogenetic tree space. A phylogenetic landscape or tree space refers to the entire combinatorial space comprising all possible phylogenetic tree topologies for a set of n taxa. The landscape of n taxa can be defined as consisting of a finite set T of tree topologies. Tree topologies can be associated with a fitness function $f(t_i)$ describing their fit. This forms a discrete solution search space and finite graph $(T, E) = G$. $E(G)$ refers to the neighborhood relation on $T(G)$. Edges in this graph are bidirectional and represent transformation from one tree topology to another by a tree rearrangement operator. An edge between t_i and t_j would be notated as e_{ij} in $E(G)$.

```

class pylogeny.landscape.graph (gr=None)
    Bases: object

    Define an empty graph object.

    __init__ (gr=None)
        Instantiate a graph.

        Parameters gr – A networkx graph object, if already exists.

    clearEdgeWeights ()

    getCenter ()
        Get the centre of the graph.

```

getCliqueNumber ()
Get the clique number of the graph.

getCliques ()
Get the cliques present in the graph.

getCliquesOfNode (*i*)
Get the clique that a node corresponds to.

GetComponentOfNode (*i*)
Get the graph component of a given node.

getComponents ()
Get the connected components in the graph.

getDegreeFor (*i*)
Return in- and out-degree for node named *i*.

getDiameter ()
Acquire the diameter of the graph.

getEdge (*i, j*)

getEdges ()

getEdgesFor (*i*)

getMST ()
Acquire the minimum spanning tree for the graph.

getNeighborsFor (*i*)

getNode (*i*)

getNodeNames ()
Return the names of nodes in the graph.

getNodes ()

getNumCliques ()
Get the number of cliques found in the graph.

getNumComponents ()
Get the number of components of the graph.

getShortestPath (*nodA, nodB*)
Get the shortest path between two nodes.

getShortestPathLength (*nodA, nodB*)
Get the shortest path length between two nodes.

getSize ()
Return the number of nodes in the graph.

hasPath (*nodA, nodB*)
See if a path exists between two nodes.

isEdge (*i, j*)

iterNodes ()
Iterate over all node keys.

setDefaultWeight (*w*)

class `pylogeny.landscape.landscape` (*ali*, *starting_tree=None*, *root=True*, *operator='SPR'*)

Bases: `pylogeny.landscape.graph`, `pylogeny.tree.treeSet`

Defines an entire phylogenetic tree space.

Parameters

- **ali** – An `alignment.alignment` object.
- **starting_tree** – An optional tree object to start

the landscape with. :param root: Whether or not to acquire an approximate maximum likelihood tree (FastTree) or start the landscape with a given starting tree. :param operator: A string that describes what operator the landscape is mostly comprised of.

__init__ (*ali*, *starting_tree=None*, *root=True*, *operator='SPR'*)

addTree (*tr*, *score=True*, *check=True*, *newick=None*, *struct=None*)

Add a tree to the landscape. Will return its index.

addTreeByNewick (*newick*, *score=True*, *check=True*, *struct=None*)

Add tree to the landscape by Newick string. Will return index.

exploreRandomTree (*i*, *type=1*)

Acquire a single neighbor to a tree in the landscape by performing a random rearrangement of type SPR (by default), NNI, or TBR – this is done by performing a rearrangement on a random branch in the topology. Rearrangement type is provided as a rearrangement module type definition of form, for example, TYPE_SPR, TYPE_NNI, etc.

exploreTree (*i*, *type=1*)

Get all neighbors to a tree named *i* in the landscape using a respective rearrangement operator as defined in the rearrangement module. Rearrangement type is provided as a rearrangement module type definition of form, for example, TYPE_SPR, TYPE_NNI, etc. By default, this is TYPE_SPR.

findTree (*newick*)

Find a tree by Newick string, taking into account branch lengths. Returns the index of this tree in the landscape.

findTreeTopology (*newick*)

Find a tree by topology, not taking into account branch lengths.

findTreeTopologyByStructure (*struct*)

Find a tree by topology, not taking into account branch lengths, given the topology.

getAlignment ()

Acquire the alignment object associated with this space.

getAllPathsOfBestImprovement ()

Return all paths of best improvement as a dictionary.

getBestImprovement (*i*)

For a tree in the landscape, investigate neighbors to find a tree that leads to the best improvement of fitness function score on the basis of likelihood.

getBipartitionFoundInTreeByIndex (*tr*, *brind*, *topol=None*)

Given a tree node and a branch index, return the associated bipartition.

getGlobalOptimum ()

Get the global optimum of the current space.

getLocalOptima ()

Get all trees in the landscape that can be labelled as a local optimum.

getLocks ()

getNumberTaxa ()

Return the number of different taxa present in any respective tree in the landscape.

getPathOfBestImprovement (i)

For a tree in the landscape, investigate neighbors iteratively until a best path of score improvement is found on basis of likelihood.

getPossibleNumberRootedTrees ()

Assuming all of the trees in the space are rooted, return the maximum possible number of unrooted trees that can possibly be generated for the number of taxa of trees in the landscape.

getPossibleNumberUnrootedTrees ()

Assuming all of the trees in the space are unrooted, return the maximum possible number of unrooted trees that can possibly be generated for the number of taxa of trees in the landscape.

getRoot ()

Returns the index to the root (starting) tree of the space.

getRootTree ()

Acquire the first tree that was placed in this space.

getTree (i)

Get the object for a tree by its name.

getVertex (i)

Acquire a vertex object from the landscape; this is a high-level representation of a tree in the landscape with additional functionality. Object created upon invocation of this function.

indexOf (tr)

Acquire the index/name in this landscape of a tree object. Returns -1 if not found.

isLocalOptimum (i)

Determine if a tree is a local optimum. This means it has the following properties:

1. Possesses a likelihood score.
2. Local neighborhood completely enumerated (and scored).
3. None of its neighbors is a better improvement.

isViolating (i)

Determine if a tree is violating any locks intrinsic to the landscape.

iterAllPathsOfBestImprovement ()

Return an iterator for all paths of best improvement.

lockBranchFoundInTree (tr, br)

Given a tree node and a branch object, add a given bipartition to the bipartition lock list. Returns bipartition if locked.

lockBranchFoundInTreeByIndex (tr, brind)

Given a tree node and a branch index, add an associated bipartition to the bipartition lock list. Returns the bipartition if locked.

removeTree (tree)

Remove a tree from the landscape by object.

removeTreeByIndex (i)

Remove a tree from the landscape by index.

setAlignment (ali)

Set the alignment present in this landscape. WARNING; will not modify existing scores.

setOperator (*op*)

Set the operator assigned to this landscape.

toProperNewickTreeSet ()

Convert this landscape into an unorganized set of trees where taxa names are transformed to their original form (i.e. not transformed to a state friendly for the Phylip format).

toTreeSet ()

Convert this landscape into an unorganized set of trees.

toggleLock (*lock*)

Add a bipartition to the list of locked bipartitions if not present; otherwise, remove it. Return status of lock.

class `pylogeny.landscape.vertex` (*obj, ls*)

Bases: `object`

Encapsulate a single vertex in the landscape and add convenient functionality to alias parent landscape functions.

__init__ (*obj, ls*)

approximatePossibleNumNeighbors ()

Approximate the possible number of neighbors to this vertex by considering the type of tree rearrangement operator.

getBestImprovement ()

Alias function for function of same name in parent landscape.

getBipartitionScores ()

Get all corresponding bipartition vectors of SPR scores.

getBipartitions ()

Get all bipartitions for this vertex.

getDegree ()

getDict ()

getIndex ()

getNeighbors ()

getNeighborsOfBipartition (*bi*)

Get corresponding neighbors of a bipartition in this vertex's tree.

getNeighborsOfBranch (*br*)

Get corresponding neighbors of a branch in this vertex's tree.

getNewick ()

getObject ()

getOrigin ()

getPathOfBestImprovement ()

Alias function for function of same name in parent landscape.

getProperNewick ()

Get the proper Newick string for a tree. :returns: A string.

getScore ()

getTree ()

isBestImprovement ()

Check to see if this vertex is a best move for another node.

isExplored()
isFailed()
isLocalOptimum()
isViolating()
Alias function for function of same name in parent landscape.
iterBipartitions()
Return a generator to iterate over all bipartitions for this vertex.
scoreLikelihood()
Acquire the log-likelihood for this vertex.
setExplored(*exp*)
Sets the “explored” flag of this node in the landscape.

1.1.8 pylogeny.landscapeWriter module

Serialize a phylogenetic landscape into an SQLite database file made up of three components: all tree IDs and respective scores, the alignment file as a set of sequences, and a representation of the graph as an edge list.

class `pylogeny.landscapeWriter.landscapeParser(path)`
Bases: `object`
Encapsulates the construction of a landscape object from a sqlite landscape file.
__init__(*path*)
getName()
Acquire the name of the parsed landscape.
parse()
Parse the file.
class `pylogeny.landscapeWriter.landscapeWriter(landscape, name)`
Bases: `object`
Encapsulate the writing of a landscape to a file format.
__init__(*landscape, name*)
writeFile(*path*='.')
Write the landscape serialized file to given path.

1.1.9 pylogeny.model module

Phylogenetic tree scoring models; intended to be coupled with the use of pybeaglehon (BEAGLE) high-performance library.

class `pylogeny.model.DiscreteStateModel(alignment)`
Bases: `object`
Initialize a discrete state model for phylogenetic data. State frequencies and character time are determined from the given alignment object.
__init__(*alignment*)
getAlignment()
getAlignmentAsStateList()

```

getCharType ()
getFrequencyOfState (i)
getRawFrequencyOfState (i)
getRawStateFreqs ()
getRawStateFreqsAsDict ()
getRawStateFreqsAsList ()
getSequenceMatrix ()
getStateFreqs ()
exception pylogeny.model.PhyloModelError (v)
    Bases: exceptions.Exception
    __init__ (v)

```

1.1.10 pylogeny.newick module

Newick string parsing and object interaction. A Newick string can represent a phylogenetic tree.

```

exception pylogeny.newick.ParsingError (val)
    Bases: exceptions.Exception
    __init__ (val)

pylogeny.newick.assignParents (top)
    Should be a one-time use function. Goes through and assigns parents to the parsed newick tree structure nodes
    and branches to allow for up-traversal.

class pylogeny.newick.branch (chi, l, parent=None, s=None)
    Bases: pylogeny.base.treeBranch
    Newick branch.
    __init__ (chi, l, parent=None, s=None)

pylogeny.newick.getAllBranches (br)
    Given a branch, traverse subtree and return comprising branches as a list.

pylogeny.newick.getBalancingBracket (newick, i)
    Given a position of an opening bracket in a newick string, i, output the closing bracket's position that corresponds
    to this opening bracket.

pylogeny.newick.getBranchLength (newick, i)
    Given a position of a colon symbol (indicating a branch length), return the branch length.

pylogeny.newick.getLeafName (newick, i)
    Given the position of a leaf, find its complete name.

pylogeny.newick.invertAlongPathToNode (target, top)
    DANGEROUS: Reverses all directionality to a given node from a top-level node. Intended as a low-level
    function for rerooting a tree.

pylogeny.newick.isSibling (br, other)
    Given a branch, determine if that branch is adjacent to another branch.

class pylogeny.newick.newickParser (newick)
    Parsing object for Newick strings.
    __init__ (newick)

```

```
parse ()
    Parse the stored newick string into a topological structure.

class pylogeny.newick.node (lbl='', children=None, parent=None)
    Bases: pylogeny.base.treeNode
    Newick node.

    __init__ (lbl='', children=None, parent=None)

pylogeny.newick.parseNewick (newick, i, j, top)
    Parse a newick string into a topological newick structure given a top-level node.

pylogeny.newick.removeBranchLengths (top)
    Goes through and removes any stored branch lengths.

pylogeny.newick.removeUnaryInternalNodes (top)
    Goes through and ensures any degree-2 internal nodes are smoothed into a single degree-3 internal node.

pylogeny.newick.shuffleLeaves (top)
    DANGEROUS: Given a top-level node, shuffle all leaves in this tree.
```

1.1.11 pylogeny.parsimony module

Toolkit for performance of Parsimonious Criterion (Parsimony) methods of optimization of a phylogenetic topology with a particular set of data.

```
pylogeny.parsimony.fitch (topology, alignment)
    Perform the Fitch algorithm on a given tree topology and associated alignment. Deprecated: Python implementation of the Fitch algorithm; see fitch C++ module for a C++ implementation that is roughly four times faster.

pylogeny.parsimony.fitch_cost (topology, profiles)
    Calculate the cost using Fitch algorithm on profile set and alignment. Deprecated: Python implementation of the Fitch algorithm; see fitch C++ module for a C++ implementation that is roughly four times faster.

class pylogeny.parsimony.profile_set (alignment)
    Hold a set of site_profile profiles for an entire alignment.

    __init__ (alignment)

    get (val)

    getForTaxa (val, tax)

    weight (val)

class pylogeny.parsimony.site_profile (alignment, site)
    Consolidate the single-column alignment at a region into a set of components on the basis of similarity alone.

    __init__ (alignment, site)
```

1.1.12 pylogeny.pll module

Wrap C extension for libpll library for use in natural Python.

```
class pylogeny.pll.dataModel (topo, alignm, model=None)
    Encapsulating a phylogenetic tree (as topology) + corresponding alignment into a libpll-associated data structure. Allows for log-likelihood scoring of this model. MUST BE CLOSED AFTER USE.
```

```
__init__ (topo, alignm, model=None)
    Initialize the data model and respective structures.
```

Parameters

- **topo** – A topology object.
- **alignm** – A `phylypFriendlyAlignment` object.

```
close ()
    If done with this particular problem. Frees associated memory.
```

```
getLogLikelihood ()
    Calculates log-likelihood using libpll.
```

```
getNewickString ()
    Acquire the Newick string of the problem instance.
```

```
class pylogeny.pll.partitionModel (ali)
    A partition model intended for libpll.
```

```
__init__ (ali)
```

```
close ()
    Delete file.
```

```
createModel (models, partnames, ranges)
    Establish a more complex model.
```

```
createSimpleModel (protein, pmodel='WAG')
    Establish a simple model (e.g., one type).
```

```
getFileName ()
    Get the file name of the model file.
```

1.1.13 pylogeny.rearrangement module

Phylogenetic tree structure encapsulation; allow rearrangement of said structure. Tree rearrangements inducing other topologies include Nearest Neighbor Interchange (NNI), Subtree Pruning and Regrafting (SPR), and Tree Bisection and Reconstruction (TBR). Each of these describe a transfer of one node in phylogenetic trees from one parent of a tree to a new parent. Respectively, these operators describe transformations that are subsets of those possible by the successive operator. For example, an NNI operator can perform transformations that are a subset of the transformations possible by the SPR operator.

```
exception pylogeny.rearrangement.RearrangementError (val)
    Bases: exceptions.Exception
```

```
__init__ (val)
```

```
pylogeny.rearrangement.dup (topo, where=None)
```

```
class pylogeny.rearrangement.rearrangement (struct, type, targ, dest)
    Encapsulates a single rearrangement move of type SPR, NNI, ...
```

```
__init__ (struct, type, targ, dest)
    Initialize by providing a pointer to a base topology, a target branch to be moved, and its destination.
```

```
doMove ()
```

```
getType ()
    Get the type of movement.
```

```
isNNI ()
```

isSPR()

isTBR()

toNewick()

Commit the move but do not create a new structure. Only retrieve resultant Newick string; will be more efficient.

toTopology()

Commit the actual move and return the topology.

toTree()

Commit the move and transform to tree object.

class `pylogeny.rearrangement.topology` (*t=None, rerootToLeaf=True, toLeaf=None*)

Bases: `pylogeny.base.treeStructure`

Encapsulate a tree topology, wrapping the newick tree structure. Is immutable.

NNI (*branch, destination*)

Perform an NNI move of a branch to a destination, only if that destination branch is a parent's parent or a parent's sibling. Returns a rearrangement structure (not the actual new structure) that can then be polled for the actual move; this is in order to save memory.

SPR (*branch, destination*)

Perform an SPR move of a branch to a destination branch, creating a new node there. Returns a rearrangement structure (not the actual new structure) that can then be polled for the actual move; this is in order to save memory.

__init__ (*t=None, rerootToLeaf=True, toLeaf=None*)

Initialize structure with a top-level internal node OR nothing.

allNNI()

Consider all valid NNI moves for a given topology and return all possible rearrangements.

allNNIForBranch (*br, flip=True*)

Consider all valid NNI moves for a given branch in the topology and return all possible rearrangements.

allSPR()

Consider all valid SPR moves for a given topology and return all possible rearrangements.

allSPRForBranch (*br, flip=True*)

Consider all valid SPR moves for a given branch in the topology and return all possible rearrangements.

allType (*type=1*)

Consider all valid moves of a given rearrangement operator for a given topology. Uses a given rearrangement operator type defined in this module. For example, calling this function by providing TYPE_NNI as the type will iterate over all NNI operations. By default, the type is TYPE_SPR.

fromNewick (*newickstr*)

Alias for `parse()`.

getBipartitions()

Get all bipartitions.

getBranchFromBipartition (*bip*)

Given a bipartition object, return a branch that creates that partition of taxa.

getBranchFromStrBipartition (*bip*)

Given a bipartition of taxa, return a branch that creates that partition of tree taxa.

getBranches()

getLeaves()

getStrBipartitionFromBranch (*br*)

Given a branch, return corresponding bipartition.

iterNNIForBranch (*br*, *flip=True*)

Consider all valid NNI moves for a given branch in the topology and yield all possible rearrangements as a generator.

iterSPRForBranch (*br*, *flip=True*)

Consider all valid SPR moves for a given branch in the topology and yield all possible rearrangements as a generator.

iterTypeForBranch (*br*, *type=1*, *flip=True*)

Iterate over all possible rearrangements for a branch using a given rearrangement operator type defined in this module. For example, calling this function by providing TYPE_NNI as the type will iterate over all NNI operations. By default, the type is TYPE_SPR.

lockBranch (*branch*)

Given a branch, lock it such that no transitions can ever occur across it.

move (*branch*, *destination*, *returnStruct=True*)

Move a branch and attach to a destination branch. Return new structure, or return merely the resultant Newick string.

parse (*newickstr*)

Parse a newick string and assign the tree to this object. Cannot already be initialized with a tree.

rerootToLeaf (*toleaf=None*)

PRIVATE: Reroots the given tree structure such that it is rooted nearest the lowest-order leaf.

toNewick ()

Return the newick string of the tree.

toTree ()

Return the tree object for this topology.

toUnrootedNewick ()

Return the newick string of the tree as an unrooted topology with a multifurcating top-level node.

toUnrootedTree ()

Return the tree object of the unrooted version of this topology.

1.1.14 pylogeny.scoring module

Functions for phylogenetic tree goodness-of-fit scoring.

`pylogeny.scoring.beaglegetLogLikelihood` (*tree*, *alignment*)

Acquire log-likelihood via C++ library BEAGLE via use of pybeaglethon wrapper library. Currently uses HKY85 model.

Parameters

- **tree** – A tree object.
- **alignment** – An alignment object.

Returns A floating point value.

`pylogeny.scoring.getLogLikelihood` (*tree*, *alignment*, *updateBranchLengths=True*)

Acquire log-likelihood via C library libpll.

Parameters

- **tree** – A tree object.

- **alignment** – An alignment object.
- **updateBranchLengths** – Whether or not to update the branch lengths

in the provided tree with optimized ones. :returns: A floating point value.

`pylogeny.scoring.getParsimony(newick, alignment)`

Acquire parsimony via a C++ implementation.

Parameters

- **newick** – A New Hampshire (Newick) tree string.
- **alignment** – An alignment object.

Returns An integer value.

`pylogeny.scoring.getParsimonyForTopology(topo, alignment)`

Acquire parsimony via a C++ implementation.

Parameters

- **topo** – A topology object.
- **alignment** – An alignment object.

Returns An integer value.

`pylogeny.scoring.getParsimonyFromProfiles(newick, profiles)`

Acquire parsimony via a C++ implementation.

Parameters

- **newick** – A New Hampshire (Newick) tree string.
- **profiles** – A set of profiles corresponding to an alignment.

Returns An integer value.

`pylogeny.scoring.getParsimonyFromProfilesForTopology(topology, profiles)`

Acquire parsimony via a C++ implementation.

Parameters

- **topo** – A topology object.
- **profiles** – A set of profiles corresponding to an alignment.

Returns An integer value.

1.1.15 pylogeny.tree module

Container definition for (phylogenetic) bifurcating or multifurcating trees defined using Newick strings, collections of them, and for splits of these trees.

class `pylogeny.tree.bipartition(topol, bra=None)`

Bases: object

A tree bipartition. Requires a tree topology. Using the term borrowed from nomenclature of a bipartite graph, a bipartition for a phylogenetic tree coincides with the definition of two disjoint sets U and V . A branch in a phylogenetic tree defines a single bipartition that divides the tree into two disjoint sets U and V . The set U comprises all of the children leaf of the subtree associated with that branch. The set V contains the rest of the leaves or taxa in the tree.

`__init__` (*topol, bra=None*)

Construct a bipartition from a branch in a topology.

Parameters

- **topol** – A topology.
- **bra** – An optional argument; can still acquire a bipartition from a

string. :type bra: :class: *newick.branch*

fromStringRepresentation (*st*)

Acquire all component elements from a string representation of a bipartition.

Parameters **st** – A string representation from a *bipartition* object.

getBestSPRScore (*ls, node=None*)

Given a landscape, return the best SPR score.

getBranch ()

Get branch corresponding to this bipartition.

Returns *newick.branch*

getBranchIndex ()

Return an index of the branch with respect to a post order traversal of the topology.

Returns an integer

getBranchListRepresentation ()

Get the tuple of lists of branches that represent this bipartition.

getMedianSPRScore (*ls, node=None*)

Given a landscape, return the median SPR score.

getSPRRearrangements ()

Return the set of all scores related to this bipartition.

getSPRScores (*ls, node=None*)

Given a landscape, return all possible scores, not actively performing scoring if not done.

getShortStringMappings ()

Get the mapping of symbols from taxa names for the shorter string representation.

getShortStringRepresentation ()

Get the shorter string representation corresponding to this bipartition.

Returns a string

getStringRepresentation ()

Get the string representation corresponding to this bipartition.

Returns a string

pylogeny.tree.numberRootedTrees (*t*)

pylogeny.tree.numberUnrootedTrees (*t*)

class *pylogeny.tree.tree* (*newi='', check=False, structure=None*)

Bases: *object*

Defines a single (phylogenetic) tree by newick string; can possess other metadata.

__init__ (*newi='', check=False, structure=None*)

If enabled, “check” will force the structure to reroot the given Newick string tree to a lowest-order leaf in order to ensure a consistent Newick string among any duplicate topologies. If a structure is provided and check is disabled, all parsing routines are bypassed and the Newick and Structure fields of this tree are overridden by the appropriate arguments.

Parameters **newi** – A Newick or New Hampshire string for a tree (unrooted or rooted). :type newi: string :param check: An optional argument; boolean indicating whether to perform parsing checks on the string. May change what Newick string is assigned without changing tree topology.

getName ()

Gets the name of this tree if it has been defined.

Returns a string

getNewick ()

Gets the Newick (New Hampshire) string for this tree.

Returns a string

getOrigin ()

Gets the “origin” of this tree, or where this tree was acquired or constructed from. Usually set by other code or an interface.

Returns string or None

getRerootedNoBranchLengthNewick ()

Returns the tree’s “structure”, a Newick string without any branch lengths.

Returns a string

getScore ()

Gets the score(s) (objective function) for this tree if it/they has/have been defined.

Returns a tuple of floats or integers

getSimpleNewick ()

Return a Newick string with all taxa name replaced with successive integers.

Returns a string

getStructure ()

Returns the tree’s “structure”, a Newick string without any branch lengths.

Returns a string

setName (*n*)

Sets the name of this tree (object).

Parameters **n** (*a string*) – A string indicating this tree’s name.

setOrigin (*o*)

Set the “origin” or specification of where this tree was acquired or constructed from.

Parameters **o** (*string or None*) – A string indicating where the tree came from.

setScore (*s*)

Sets the score(s) for this tree. Should be performed by a scorer (see scoring functions in the appropriate module).

Parameters **s** (*a tuple of floats or integers*) – A set of objective function scores.

toNewick ()

Gets the Newick (New Hampshire) string for this tree.

Returns a string

toTopology ()

Return a topology object instance for this tree to allow for rearrangement of the actual structure of the tree.

Returns a :class: *rearrangement.topology* object

updateNewick (*n*, *reroot=False*)

Update the contained Newick string only as long as the structure obtained (after rerooting, which is an optional parameter) is identical to the contained structure.

Parameters

- **n** (*a string*) – A Newick or New Hampshire formatted string.
- **reroot** – A boolean indicating whether to reroot the provided

Newick string to a lexicographically lowest-order taxa name to ensure redundant topologies across other trees.

class `pylogeny.tree.treeSet`

Bases: `_abcoll.Sized`, `_abcoll.Iterable`

Represents an ordered, disorganized collection of trees that do not necessarily comprise a combinatorial space.

__init__ ()

addTree (*tr*)

Add a tree object to the collection.

Parameters **tr** – A tree object.

addTreeByNewick (*newick*)

Add a tree to the structure by Newick string.

Parameters **newick** (*a string*) – A New Hampshire or Newick string.

static fromTreeFile (*fin*)

Acquire a file where newlines separate Newick strings, and create an instance of treeSet from those trees.

indexOf (*tr*)

Acquire the index in this collection of a tree object. Returns -1 if not found.

Parameters **tr** – A tree object.

Returns an integer [-1,length of collection)

iterTrees ()

Iterate over all trees found in this set.

removeTree (*tr*)

Remove a tree object from the collection if present.

Parameters **tr** – A tree object (present in the collection).

toTreeFile (*fout*)

Output this landscape as a series of trees, separated by newlines, as a text file saved at the given path.

Parameters **fout** (*a string*) – A string indicating a file system path to a file.

1.2 Module contents

Pylogeny is a Python library and code framework for phylogenetic tree reconstruction and scoring.

Allows one to perform the following tasks: (1) Generate and manage phylogenetic tree landscapes. (2) Build and rearrange phylogenetic trees using preset operators such as NNI, SPR, and TBR. (3) Score phylogenetic trees by Log-likelihood and Parsimony.

Dependencies: Pandas, P4 Phylogenetic Library. Suggested: FastTree, RAxML, PytBEAGLEhon.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

p

- pylogeny, [25](#)
- pylogeny.alignment, [3](#)
- pylogeny.base, [5](#)
- pylogeny.database, [7](#)
- pylogeny.executable, [8](#)
- pylogeny.heuristic, [10](#)
- pylogeny.JSONWriter, [3](#)
- pylogeny.landscape, [11](#)
- pylogeny.landscapeWriter, [16](#)
- pylogeny.model, [16](#)
- pylogeny.newick, [17](#)
- pylogeny.parsimony, [18](#)
- pylogeny.pll, [18](#)
- pylogeny.rearrangement, [19](#)
- pylogeny.scoring, [21](#)
- pylogeny.tree, [22](#)

Symbols

- `__init__()` (pylogeny.JSONWriter.JSONWriter method), 3
 - `__init__()` (pylogeny.alignment.alignment method), 3
 - `__init__()` (pylogeny.alignment.phylipFriendlyAlignment method), 4
 - `__init__()` (pylogeny.base.treeBranch method), 5
 - `__init__()` (pylogeny.base.treeNode method), 5
 - `__init__()` (pylogeny.base.treeStructure method), 6
 - `__init__()` (pylogeny.base.trie method), 6
 - `__init__()` (pylogeny.database.SQLiteDatabase method), 7
 - `__init__()` (pylogeny.database.SQLExhaustiveLandscape method), 7
 - `__init__()` (pylogeny.database.SQLiteDatabase method), 7
 - `__init__()` (pylogeny.database.SQLiteLandscape method), 8
 - `__init__()` (pylogeny.executable.aTemporaryDirectory method), 8
 - `__init__()` (pylogeny.executable.consel method), 9
 - `__init__()` (pylogeny.executable.fasttree method), 9
 - `__init__()` (pylogeny.executable.raxml method), 9
 - `__init__()` (pylogeny.executable.rspr method), 10
 - `__init__()` (pylogeny.executable.treepuzzle method), 10
 - `__init__()` (pylogeny.heuristic.RAxMLIdentify method), 10
 - `__init__()` (pylogeny.heuristic.heuristic method), 10
 - `__init__()` (pylogeny.heuristic.likelihoodGreedy method), 10
 - `__init__()` (pylogeny.heuristic.parsimonyGreedy method), 11
 - `__init__()` (pylogeny.heuristic.phylogeneticLinearHeuristic method), 11
 - `__init__()` (pylogeny.heuristic.smoothGreedy method), 11
 - `__init__()` (pylogeny.landscape.graph method), 11
 - `__init__()` (pylogeny.landscape.landscape method), 13
 - `__init__()` (pylogeny.landscape.vertex method), 15
 - `__init__()` (pylogeny.landscapeWriter.landscapeParser method), 16
 - `__init__()` (pylogeny.landscapeWriter.landscapeWriter method), 16
 - `__init__()` (pylogeny.model.DiscreteStateModel method), 16
 - `__init__()` (pylogeny.model.PhyloModelError method), 17
 - `__init__()` (pylogeny.newick.ParsingError method), 17
 - `__init__()` (pylogeny.newick.branch method), 17
 - `__init__()` (pylogeny.newick.newickParser method), 17
 - `__init__()` (pylogeny.newick.node method), 18
 - `__init__()` (pylogeny.parsimony.profile_set method), 18
 - `__init__()` (pylogeny.parsimony.site_profile method), 18
 - `__init__()` (pylogeny.pll.dataModel method), 18
 - `__init__()` (pylogeny.pll.partitionModel method), 19
 - `__init__()` (pylogeny.rearrangement.RearrangementError method), 19
 - `__init__()` (pylogeny.rearrangement.rearrangement method), 19
 - `__init__()` (pylogeny.rearrangement.topology method), 20
 - `__init__()` (pylogeny.tree.bipartition method), 22
 - `__init__()` (pylogeny.tree.tree method), 23
 - `__init__()` (pylogeny.tree.treeSet method), 25
- ## A
- `addChild()` (pylogeny.base.treeNode method), 5
 - `addTree()` (pylogeny.landscape.landscape method), 13
 - `addTree()` (pylogeny.tree.treeSet method), 25
 - `addTreeByNewick()` (pylogeny.landscape.landscape method), 13
 - `addTreeByNewick()` (pylogeny.tree.treeSet method), 25
 - `alignment` (class in pylogeny.alignment), 3
 - `allNNI()` (pylogeny.rearrangement.topology method), 20
 - `allNNIForBranch()` (pylogeny.rearrangement.topology method), 20
 - `allSPR()` (pylogeny.rearrangement.topology method), 20
 - `allSPRForBranch()` (pylogeny.rearrangement.topology method), 20
 - `allType()` (pylogeny.rearrangement.topology method), 20
 - `alphabet` (pylogeny.base.trie attribute), 6
 - `approximatePossibleNumNeighbors()` (pylogeny.landscape.vertex method), 15
 - `assignParents()` (in module pylogeny.newick), 17
 - `aTemporaryDirectory` (class in pylogeny.executable), 8
- ## B
- `beaglegetLogLikelihood()` (in module pylogeny.scoring),

21

bestTree (pylogeny.heuristic.phylogeneticLinearHeuristic attribute), 11

bipartition (class in pylogeny.tree), 22

bootstrap() (pylogeny.alignment.alignment method), 3

branch (class in pylogeny.newick), 17

C

child (pylogeny.base.treeBranch attribute), 5

children (pylogeny.base.treeNode attribute), 5

clearEdgeWeights() (pylogeny.landscape.graph method), 11

close() (pylogeny.alignment.alignment method), 3

close() (pylogeny.database.database method), 8

close() (pylogeny.database.SQLDatabase method), 7

close() (pylogeny.database.SQLiteDatabase method), 7

close() (pylogeny.pll.dataModel method), 19

close() (pylogeny.pll.partitionModel method), 19

connect() (pylogeny.database.SQLDatabase method), 7

consel (class in pylogeny.executable), 8

count (pylogeny.base.trie attribute), 6

createModel() (pylogeny.pll.partitionModel method), 19

createSimpleModel() (pylogeny.pll.partitionModel method), 19

cursor (pylogeny.database.database attribute), 8

D

database (class in pylogeny.database), 8

DatabaseLandscape (class in pylogeny.database), 7

dataModel (class in pylogeny.pll), 18

delete() (pylogeny.base.patriciaTree method), 5

delete() (pylogeny.base.trie method), 6

DiscreteStateModel (class in pylogeny.model), 16

doMove() (pylogeny.rearrangement.rearrangement method), 19

dup() (in module pylogeny.rearrangement), 19

E

executable (class in pylogeny.executable), 9

exeExists() (in module pylogeny.executable), 9

exeName (pylogeny.executable.executable attribute), 9

exeName (pylogeny.executable.fasttree attribute), 9

exeName (pylogeny.executable.raxml attribute), 9

exeName (pylogeny.executable.rspr attribute), 10

exeName (pylogeny.executable.treepuzzle attribute), 10

explore() (pylogeny.heuristic.heuristic method), 10

explore() (pylogeny.heuristic.likelihoodGreedy method), 10

explore() (pylogeny.heuristic.parsimonyGreedy method), 11

explore() (pylogeny.heuristic.RAxMLIdentify method), 10

explore() (pylogeny.heuristic.smoothGreedy method), 11

exploreRandomTree() (pylogeny.database.SQLExhaustiveLandscape method), 7

exploreRandomTree() (pylogeny.landscape.landscape method), 13

exploreTree() (pylogeny.database.SQLExhaustiveLandscape method), 7

exploreTree() (pylogeny.landscape.landscape method), 13

F

fasttree (class in pylogeny.executable), 9

filterRecords() (pylogeny.database.database method), 8

findTree() (pylogeny.landscape.landscape method), 13

findTreeTopology() (pylogeny.landscape.landscape method), 13

findTreeTopologyByStructure() (pylogeny.landscape.landscape method), 13

fitch() (in module pylogeny.parsimony), 18

fitch_cost() (in module pylogeny.parsimony), 18

fromNewick() (pylogeny.rearrangement.topology method), 20

fromStringRepresentation() (pylogeny.tree.bipartition method), 23

fromTreeFile() (pylogeny.tree.treeSet static method), 25

G

get() (pylogeny.parsimony.profile_set method), 18

getAlignment() (pylogeny.alignment.alignment method), 3

getAlignment() (pylogeny.landscape.landscape method), 13

getAlignment() (pylogeny.model.DiscreteStateModel method), 16

getAlignmentAsStateList() (pylogeny.model.DiscreteStateModel method), 16

getAllBranches() (in module pylogeny.newick), 17

getAllLeaves() (pylogeny.base.treeStructure method), 6

getAllNodes() (pylogeny.base.treeStructure method), 6

getAllPathsOfBestImprovement() (pylogeny.landscape.landscape method), 13

getAlphabet() (pylogeny.base.trie method), 6

getApproxMLNewick() (pylogeny.alignment.alignment method), 3

getApproxMLTree() (pylogeny.alignment.alignment method), 4

getBalancingBracket() (in module pylogeny.newick), 17

getBestImprovement() (pylogeny.landscape.landscape method), 13

getBestImprovement() (pylogeny.landscape.vertex method), 15

getBestSPRScore() (pylogeny.tree.bipartition method), 23

getBestTree() (pylogeny.heuristic.phylogeneticLinearHeuristic method), 11
 getBipartitionFoundInTreeByIndex() (pylogeny.landscape.landscape method), 13
 getBipartitions() (pylogeny.landscape.vertex method), 15
 getBipartitions() (pylogeny.rearrangement.topology method), 20
 getBipartitionScores() (pylogeny.landscape.vertex method), 15
 getBranch() (pylogeny.tree.bipartition method), 23
 getBranches() (pylogeny.rearrangement.topology method), 20
 getBranchFromBipartition() (pylogeny.rearrangement.topology method), 20
 getBranchFromStrBipartition() (pylogeny.rearrangement.topology method), 20
 getBranchIndex() (pylogeny.tree.bipartition method), 23
 getBranchLength() (in module pylogeny.newick), 17
 getBranchListRepresentation() (pylogeny.tree.bipartition method), 23
 getCenter() (pylogeny.landscape.graph method), 11
 getCharType() (pylogeny.model.DiscreteStateModel method), 16
 getChild() (pylogeny.base.treeBranch method), 5
 getChildByIndex() (pylogeny.base.treeNode method), 5
 getChildren() (pylogeny.base.treeNode method), 5
 getCliqueNumber() (pylogeny.landscape.graph method), 11
 getCliques() (pylogeny.landscape.graph method), 12
 getCliquesOfNode() (pylogeny.landscape.graph method), 12
 getColumns() (pylogeny.database.database method), 8
 getColumns() (pylogeny.database.SQLDatabase method), 7
 getColumns() (pylogeny.database.SQLiteDatabase method), 7
 getCompleteLandscape() (pylogeny.JSONWriter.JSONWriter method), 3
 getComponentOfNode() (pylogeny.landscape.graph method), 12
 getComponents() (pylogeny.landscape.graph method), 12
 getDatabaseNode() (pylogeny.database.SQLExhaustiveLandscape method), 7
 getDegree() (pylogeny.landscape.vertex method), 15
 getDegreeFor() (pylogeny.landscape.graph method), 12
 getDiameter() (pylogeny.landscape.graph method), 12
 getDict() (pylogeny.landscape.vertex method), 15
 getDim() (pylogeny.alignment.alignment method), 4
 getEdge() (pylogeny.landscape.graph method), 12
 getEdges() (pylogeny.landscape.graph method), 12
 getEdgesFor() (pylogeny.landscape.graph method), 12
 getFASTA() (pylogeny.alignment.alignment method), 4
 getFastTreeNewick() (pylogeny.alignment.alignment method), 4
 getFileName() (pylogeny.pll.partitionModel method), 19
 getForTaxa() (pylogeny.parsimony.profile_set method), 18
 getFrequencyOfState() (pylogeny.model.DiscreteStateModel method), 17
 getGlobalOptimum() (pylogeny.landscape.landscape method), 13
 getHeaders() (pylogeny.database.database method), 8
 getIndex() (pylogeny.landscape.vertex method), 15
 getInstructionString() (pylogeny.executable.consel method), 9
 getInstructionString() (pylogeny.executable.executable method), 9
 getInstructionString() (pylogeny.executable.fasttree method), 9
 getInstructionString() (pylogeny.executable.raxml method), 9
 getInstructionString() (pylogeny.executable.rspr method), 10
 getInstructionString() (pylogeny.executable.treepuzzle method), 10
 getInterval() (pylogeny.executable.consel method), 9
 getJSON() (pylogeny.JSONWriter.JSONWriter method), 3
 getLabel() (pylogeny.base.treeBranch method), 5
 getLabel() (pylogeny.base.treeNode method), 5
 getLeafName() (in module pylogeny.newick), 17
 getLeaves() (pylogeny.rearrangement.topology method), 20
 getLocalOptima() (pylogeny.landscape.landscape method), 13
 getLocks() (pylogeny.landscape.landscape method), 13
 getLogLikelihood() (in module pylogeny.scoring), 21
 getLogLikelihood() (pylogeny.pll.dataModel method), 19
 getMedianSPRScore() (pylogeny.tree.bipartition method), 23
 getMST() (pylogeny.landscape.graph method), 12
 getName() (pylogeny.landscapeWriter.landscapeParser method), 16
 getName() (pylogeny.tree.tree method), 24
 getNeighbors() (pylogeny.landscape.vertex method), 15
 getNeighborsFor() (pylogeny.landscape.graph method), 12
 getNeighborsOfBipartition() (pylogeny.landscape.vertex method), 15
 getNeighborsOfBranch() (pylogeny.landscape.vertex method), 15
 getNewick() (pylogeny.landscape.vertex method), 15
 getNewick() (pylogeny.tree.tree method), 24

[getNewickString\(\)](#) (pylogeny.pll.dataModel method), [19](#)
[getNode\(\)](#) (pylogeny.database.DatabaseLandscape method), [7](#)
[getNode\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getNodeNames\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getNodes\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getNonEmptyChildrenBranches\(\)](#) (pylogeny.base.trieNode method), [6](#)
[getNonEmptyChildrenBranchLabels\(\)](#) (pylogeny.base.trieNode method), [6](#)
[getNonEmptyChildrenNodes\(\)](#) (pylogeny.base.trieNode method), [6](#)
[getNumberTaxa\(\)](#) (pylogeny.landscape.landscape method), [13](#)
[getNumCliques\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getNumComponents\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getNumSeqs\(\)](#) (pylogeny.alignment.alignment method), [4](#)
[getObject\(\)](#) (pylogeny.landscape.vertex method), [15](#)
[getOnlyImprovements\(\)](#) (pylogeny.JSONWriter.JSONWriter method), [3](#)
[getOrigin\(\)](#) (pylogeny.landscape.vertex method), [15](#)
[getOrigin\(\)](#) (pylogeny.tree.tree method), [24](#)
[getParent\(\)](#) (pylogeny.base.treeBranch method), [5](#)
[getParent\(\)](#) (pylogeny.base.trieNode method), [5](#)
[getParentNode\(\)](#) (pylogeny.base.trieNode method), [6](#)
[getParsimony\(\)](#) (in module pylogeny.scoring), [22](#)
[getParsimonyForTopology\(\)](#) (in module pylogeny.scoring), [22](#)
[getParsimonyFromProfiles\(\)](#) (in module pylogeny.scoring), [22](#)
[getParsimonyFromProfilesForTopology\(\)](#) (in module pylogeny.scoring), [22](#)
[getPath\(\)](#) (pylogeny.heuristic.phylogeneticLinearHeuristic method), [11](#)
[getPathOfBestImprovement\(\)](#) (pylogeny.landscape.landscape method), [14](#)
[getPathOfBestImprovement\(\)](#) (pylogeny.landscape.vertex method), [15](#)
[getPhylip\(\)](#) (pylogeny.alignment.phylipFriendlyAlignment method), [4](#)
[getPossibleNumberRootedTrees\(\)](#) (pylogeny.landscape.landscape method), [14](#)
[getPossibleNumberUnrootedTrees\(\)](#) (pylogeny.landscape.landscape method), [14](#)
[getPostOrderTraversal\(\)](#) (pylogeny.base.treeStructure method), [6](#)
[getProperName\(\)](#) (pylogeny.alignment.phylipFriendlyAlignment method), [4](#)
[getProperNewick\(\)](#) (pylogeny.landscape.vertex method), [15](#)
[getRawFrequencyOfState\(\)](#) (pylogeny.model.DiscreteStateModel method), [17](#)
[getRawStateFreqs\(\)](#) (pylogeny.model.DiscreteStateModel method), [17](#)
[getRawStateFreqsAsDict\(\)](#) (pylogeny.model.DiscreteStateModel method), [17](#)
[getRawStateFreqsAsList\(\)](#) (pylogeny.model.DiscreteStateModel method), [17](#)
[getRecords\(\)](#) (pylogeny.database.database method), [8](#)
[getRecordsAsDict\(\)](#) (pylogeny.database.database method), [8](#)
[getRecordsColumn\(\)](#) (pylogeny.database.database method), [8](#)
[getRerootedNoBranchLengthNewick\(\)](#) (pylogeny.tree.tree method), [24](#)
[getRoot\(\)](#) (pylogeny.base.treeStructure method), [6](#)
[getRoot\(\)](#) (pylogeny.base.trie method), [6](#)
[getRoot\(\)](#) (pylogeny.landscape.landscape method), [14](#)
[getRootTree\(\)](#) (pylogeny.landscape.landscape method), [14](#)
[getScore\(\)](#) (pylogeny.landscape.vertex method), [15](#)
[getScore\(\)](#) (pylogeny.tree.tree method), [24](#)
[getSequence\(\)](#) (pylogeny.alignment.alignment method), [4](#)
[getSequenceMatrix\(\)](#) (pylogeny.model.DiscreteStateModel method), [17](#)
[getShortestPath\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getShortestPathLength\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getShortStringMappings\(\)](#) (pylogeny.tree.bipartition method), [23](#)
[getShortStringRepresentation\(\)](#) (pylogeny.tree.bipartition method), [23](#)
[getSimpleNewick\(\)](#) (pylogeny.tree.tree method), [24](#)
[getSiteLikelihoodFile\(\)](#) (pylogeny.executable.tree puzzle method), [10](#)
[getSize\(\)](#) (pylogeny.alignment.alignment method), [4](#)
[getSize\(\)](#) (pylogeny.landscape.graph method), [12](#)
[getSPRDistance\(\)](#) (pylogeny.executable.rspr method), [10](#)
[getSPRRearrangements\(\)](#) (pylogeny.tree.bipartition method), [23](#)
[getSPRScores\(\)](#) (pylogeny.tree.bipartition method), [23](#)
[getStartState\(\)](#) (pylogeny.heuristic.heuristic method), [10](#)
[getStateFreqs\(\)](#) (pylogeny.model.DiscreteStateModel method), [17](#)
[getStateGraph\(\)](#) (pylogeny.heuristic.heuristic method), [10](#)
[getStateModel\(\)](#) (pylogeny.alignment.alignment method), [4](#)
[getStrBipartitionFromBranch\(\)](#) (pylogeny.rearrangement.topology method), [20](#)
[getStringRepresentation\(\)](#) (pylogeny.tree.bipartition

method), 23
 getStructure() (pylogeny.tree.tree method), 24
 getTables() (pylogeny.database.database method), 8
 getTables() (pylogeny.database.SQLDatabase method), 7
 getTables() (pylogeny.database.SQLiteDatabase method), 7
 getTaxa() (pylogeny.alignment.alignment method), 4
 getTaxa() (pylogeny.alignment.phylypFriendlyAlignment method), 4
 getTree() (pylogeny.landscape.landscape method), 14
 getTree() (pylogeny.landscape.vertex method), 15
 getType() (pylogeny.rearrangement.rearrangement method), 19
 getVertex() (pylogeny.landscape.landscape method), 14
 graph (class in pylogeny.landscape), 11

H

hasPath() (pylogeny.landscape.graph method), 12
 heuristic (class in pylogeny.heuristic), 10

I

indexOf() (pylogeny.landscape.landscape method), 14
 indexOf() (pylogeny.tree.treeSet method), 25
 insert() (pylogeny.base.patriciaTree method), 5
 insert() (pylogeny.base.trie method), 6
 insertRecord() (pylogeny.database.database method), 8
 insertRecords() (pylogeny.database.database method), 8
 invertAlongPathToNode() (in module pylogeny.newick), 17
 isBestImprovement() (pylogeny.landscape.vertex method), 15
 isEdge() (pylogeny.landscape.graph method), 12
 isEmpty() (pylogeny.database.database method), 8
 isExplored() (pylogeny.landscape.vertex method), 15
 isFailed() (pylogeny.landscape.vertex method), 16
 isInternalNode() (pylogeny.base.treeNode method), 5
 isLeaf() (pylogeny.base.treeNode method), 5
 isLocalOptimum() (pylogeny.landscape.landscape method), 14
 isLocalOptimum() (pylogeny.landscape.vertex method), 16
 isNNI() (pylogeny.rearrangement.rearrangement method), 19
 isSibling() (in module pylogeny.newick), 17
 isSPR() (pylogeny.rearrangement.rearrangement method), 19
 isTBR() (pylogeny.rearrangement.rearrangement method), 20
 isViolating() (pylogeny.landscape.landscape method), 14
 isViolating() (pylogeny.landscape.vertex method), 16
 iterAllPathsOfBestImprovement() (pylogeny.landscape.landscape method), 14
 iterBipartitions() (pylogeny.landscape.vertex method), 16

iterNNIForBranch() (pylogeny.rearrangement.topology method), 21
 iterNodes() (pylogeny.landscape.graph method), 12
 iterNonEmptyChildrenNodes() (pylogeny.base.trieNode method), 6
 iterRecords() (pylogeny.database.database method), 8
 iterSPRForBranch() (pylogeny.rearrangement.topology method), 21
 iterTrees() (pylogeny.tree.treeSet method), 25
 iterTypeForBranch() (pylogeny.rearrangement.topology method), 21

J

JSONWriter (class in pylogeny.JSONWriter), 3

L

label (pylogeny.base.treeBranch attribute), 5
 label (pylogeny.base.treeNode attribute), 5
 landscape (class in pylogeny.landscape), 12
 landscapeParser (class in pylogeny.landscapeWriter), 16
 landscapeWriter (class in pylogeny.landscapeWriter), 16
 leaves() (pylogeny.base.treeStructure static method), 6
 likelihoodGreedy (class in pylogeny.heuristic), 10
 lockBranch() (pylogeny.rearrangement.topology method), 21
 lockBranchFoundInTree() (pylogeny.landscape.landscape method), 14
 lockBranchFoundInTreeByIndex() (pylogeny.landscape.landscape method), 14
 longest_common_substring() (in module pylogeny.base), 5

M

move() (pylogeny.rearrangement.topology method), 21

N

newickParser (class in pylogeny.newick), 17
 newTable() (pylogeny.database.database method), 8
 nextLabel (pylogeny.base.trie attribute), 6
 NNI() (pylogeny.rearrangement.topology method), 20
 node (class in pylogeny.newick), 18
 nodes() (pylogeny.base.treeStructure static method), 6
 nodeToJSON() (pylogeny.JSONWriter.JSONWriter method), 3
 numberRootedTrees() (in module pylogeny.tree), 23
 numberUnrootedTrees() (in module pylogeny.tree), 23
 numEmptyChildrenNodes() (pylogeny.base.trieNode method), 7

P

parent (pylogeny.base.treeBranch attribute), 5
 parent (pylogeny.base.treeNode attribute), 5
 parse() (pylogeny.landscapeWriter.landscapeParser method), 16

[parse\(\)](#) (pylogeny.newick.newickParser method), 18
[parse\(\)](#) (pylogeny.rearrangement.topology method), 21
[parseNewick\(\)](#) (in module pylogeny.newick), 18
[parsimonyGreedy](#) (class in pylogeny.heuristic), 11
[ParsingError](#), 17
[partitionModel](#) (class in pylogeny.pll), 19
[path](#) (pylogeny.heuristic.phylogeneticLinearHeuristic attribute), 11
[patriciaTree](#) (class in pylogeny.base), 5
[phylipFriendlyAlignment](#) (class in pylogeny.alignment), 4
[phylogeneticLinearHeuristic](#) (class in pylogeny.heuristic), 11
[PhyloModelError](#), 17
[postOrderTraversal\(\)](#) (pylogeny.base.treeStructure static method), 6
[profile_set](#) (class in pylogeny.parsimony), 18
[pylogeny](#) (module), 25
[pylogeny.alignment](#) (module), 3
[pylogeny.base](#) (module), 5
[pylogeny.database](#) (module), 7
[pylogeny.executable](#) (module), 8
[pylogeny.heuristic](#) (module), 10
[pylogeny.JSONWriter](#) (module), 3
[pylogeny.landscape](#) (module), 11
[pylogeny.landscapeWriter](#) (module), 16
[pylogeny.model](#) (module), 16
[pylogeny.newick](#) (module), 17
[pylogeny.parsimony](#) (module), 18
[pylogeny.pll](#) (module), 18
[pylogeny.rearrangement](#) (module), 19
[pylogeny.scoring](#) (module), 21
[pylogeny.tree](#) (module), 22

Q

[query\(\)](#) (pylogeny.database.database method), 8
[query\(\)](#) (pylogeny.database.SQLDatabase method), 7
[query\(\)](#) (pylogeny.database.SQLiteDatabase method), 7
[querymany\(\)](#) (pylogeny.database.database method), 8
[querymany\(\)](#) (pylogeny.database.SQLDatabase method), 7
[querymany\(\)](#) (pylogeny.database.SQLiteDatabase method), 7

R

[raxml](#) (class in pylogeny.executable), 9
[RAXMLIdentify](#) (class in pylogeny.heuristic), 10
[rearrangement](#) (class in pylogeny.rearrangement), 19
[RearrangementError](#), 19
[reassignFromReinterpretedNewick\(\)](#) (pylogeny.alignment.phylipFriendlyAlignment method), 4
[recreateObject\(\)](#) (pylogeny.alignment.phylipFriendlyAlignment method), 4

[reinterpretNewick\(\)](#) (pylogeny.alignment.phylipFriendlyAlignment method), 4
[removeBranchLengths\(\)](#) (in module pylogeny.newick), 18
[removeTree\(\)](#) (pylogeny.landscape.landscape method), 14
[removeTree\(\)](#) (pylogeny.tree.treeSet method), 25
[removeTreeByIndex\(\)](#) (pylogeny.landscape.landscape method), 14
[removeUnaryInternalNodes\(\)](#) (in module pylogeny.newick), 18
[rerootToLeaf\(\)](#) (pylogeny.rearrangement.topology method), 21
[root](#) (pylogeny.base.treeStructure attribute), 6
[root](#) (pylogeny.base.trie attribute), 6
[rspr](#) (class in pylogeny.executable), 9
[RSPR_ALG_APPROX](#) (pylogeny.executable.rspr attribute), 9
[RSPR_ALG_BB](#) (pylogeny.executable.rspr attribute), 10
[RSPR_ALG_DEFAULT](#) (pylogeny.executable.rspr attribute), 10
[RSPR_ALG_FPT](#) (pylogeny.executable.rspr attribute), 10
[run\(\)](#) (pylogeny.executable.executable method), 9
[runFunction\(\)](#) (pylogeny.executable.raxml method), 9

S

[scoreLikelihood\(\)](#) (pylogeny.landscape.vertex method), 16
[search\(\)](#) (pylogeny.base.patriciaTree method), 5
[search\(\)](#) (pylogeny.base.trie method), 6
[setAlignment\(\)](#) (pylogeny.landscape.landscape method), 14
[setChildNode\(\)](#) (pylogeny.base.trieNode method), 7
[setDefaultWeight\(\)](#) (pylogeny.landscape.graph method), 12
[setExplored\(\)](#) (pylogeny.landscape.vertex method), 16
[setName\(\)](#) (pylogeny.tree.tree method), 24
[setOperator\(\)](#) (pylogeny.landscape.landscape method), 14
[setOrigin\(\)](#) (pylogeny.tree.tree method), 24
[setScore\(\)](#) (pylogeny.tree.tree method), 24
[shuffleLeaves\(\)](#) (in module pylogeny.newick), 18
[site_profile](#) (class in pylogeny.parsimony), 18
[smoothGreedy](#) (class in pylogeny.heuristic), 11
[SPR\(\)](#) (pylogeny.rearrangement.topology method), 20
[SQLDatabase](#) (class in pylogeny.database), 7
[SQLExhaustiveLandscape](#) (class in pylogeny.database), 7
[SQLiteDatabase](#) (class in pylogeny.database), 7
[SQLiteLandscape](#) (class in pylogeny.database), 7

T

[toggleLock\(\)](#) (pylogeny.landscape.landscape method), 15
[toNewick\(\)](#) (pylogeny.rearrangement.rearrangement method), 20
[toNewick\(\)](#) (pylogeny.rearrangement.topology method), 21

toNewick() (pylogeny.tree.tree method), 24
topology (class in pylogeny.rearrangement), 20
toProperNewickTreeSet() (pylogeny.landscape.landscape
method), 15
toStrList() (pylogeny.alignment.alignment method), 4
toTopology() (pylogeny.rearrangement.rearrangement
method), 20
toTopology() (pylogeny.tree.tree method), 24
toTree() (pylogeny.rearrangement.rearrangement
method), 20
toTree() (pylogeny.rearrangement.topology method), 21
toTreeFile() (pylogeny.tree.treeSet method), 25
toTreeSet() (pylogeny.landscape.landscape method), 15
toUnrootedNewick() (pylogeny.rearrangement.topology
method), 21
toUnrootedTree() (pylogeny.rearrangement.topology
method), 21
tree (class in pylogeny.tree), 23
treeBranch (class in pylogeny.base), 5
treeNode (class in pylogeny.base), 5
treepuzzle (class in pylogeny.executable), 10
treeSet (class in pylogeny.tree), 25
treeStructure (class in pylogeny.base), 5
trie (class in pylogeny.base), 6
trieNode (class in pylogeny.base), 6

U

updateNewick() (pylogeny.tree.tree method), 24

V

vertex (class in pylogeny.landscape), 15

W

weight() (pylogeny.parsimony.profile_set method), 18
writeFile() (pylogeny.landscapeWriter.landscapeWriter
method), 16
writeProperNexus() (py-
logeny.alignment.phylipFriendlyAlignment
method), 4