Supplementary Material for Utilizing SVDquartets in Improving ASTRAL on High Gene Tree Estimation Error Model Conditions

Akhil Jakatdar*[1] and Sarthak Mishra†[1]

[1]Department of Computer Science, University of Illinois at Urbana-Champaign

1 Introduction

Included in the Supplementary Material for Utilizing SVDquartets in Improving ASTRAL on High Gene Tree Estimation Error Model Conditions are the commands and version numbers of all tools used in running the experiments for the paper. Additionally, all important programs and scripts that utilize these commands are need to replicate the studies included in this paper are included in this document. However, specific data files can be found in the Github listed in this document. For any additional information, feel free to email either author at the following emails: Akhil Jakatdar at akhilrj2@illinois.edu and Sarthak Mishra at mishra20@illinois.edu. The data used for this experiment was generated in the paper, Molloy & Warnow [1].

2 Commands & Dataset

2.0.1 Commands and Versions

ASTRAL-III [2]
Version **:** 5.7.8
Input **:** set of gene trees
Output **:** species Trees with branch support
Command **:** `ASTRAL -i <input gene trees> -o <output file>`
Input file can be found in: $https://github.com/smishra677/CS581\_ASTRAL\_SVD/tree/main/Rax\_ML//RAxML\_6\_[50, 100, 250, 500, 1000].tre$

RAxML [3]
Version **:** 8.2.12
Command **:** `RAxML -m GTRGAMMA -p[seed=12345] -s <path to input file>`
Input file can be found in: $https://github.com/smishra677/CS581\_ASTRAL\_SVD/tree/main//input//25tax-1000gen-0bps-500K-1E-6-rand//[04, 05, 07, 11, 13, 16, 17, 20]$

*akhilrj2@illinois.edu
†mishra20@illinois.edu

PAUP* [4]
Version **:** 4a
Input **:** sequence alignment in Nexus format
Output **:** Species Tree using SVDquartet
Command **:**
`exe <input file>;svd showScores=no evalQuartets=all qformat=qmc replace=no;savetrees file='`$out\_svd\_[50, 100, 250, 500, 1000]\_[04, 05, 07, 11, 13, 16, 17, 20].tre'$ `format='newick' replace='yes';quit;`
Input file can be found in : $https://github.com/smishra677/CS581\_ASTRAL\_SVD/tree/main/svd//$
Output file can be found in : $https://github.com/smishra677/CS581\_ASTRAL\_SVD/tree/main/svd//$

Newick Utilities [5]
Version **:** N/A
Input **:** Input tree
Output **:** A tree with collapsed edges
Command **:** `nw_ed <input>  'i & b <threshold>'o > <output>`

Dendropy [6]
Version **:** 4.5.2

2.1 Dataset

The Molloy & Warnow ASTRAL-II dataset used in the experiments run for this paper can be found at http://dx.doi.org/10.5061/dryad.km24v.

3 Scripts & Further Documented Materials

3.1 Commands

3.1.1 Getting Polytomies

Input: *input*: `Location to the collapsed.tre`

Output: *'dictionary*: `the dictionary containing all the polytomies with key at the parent node and list of children as the values`

`tree: the tree object created by dendropy`

```
#get polytomies
def get_polytomies(input_):
   #detect polytomies got from dendropy


   tax = TaxonNamespace()
   tree = Tree.get(file=open(input_, 'r'),
                schema="newick",
                tree_offset=0,
```

```
                taxon_namespace=tax,
                suppress_edge_lengths=True,
                preserve_underscores=True,
                ignore_unrecognized_keyword_arguments=False)
    polytomies = []
    dictionary_={}
    for node in tree.postorder_node_iter():
       if len(node._child_nodes) > 2:
          dictionary_[node]=[]
          for children in node._child_nodes:
             dictionary_[node].append(children)



    #pprint.pprint(tree)
    return dictionary_,tree
```

3.1.2 Selecting Polytomies

Input: *tree*: The tree object returned by the Get polytomy function

*dictionary_*: Dictionary object returned by Get polytomy function
<Key: Node in tree object, Value: List of Node objects in a polytomy>

Degree: Number of nodes to be selected from each subtree

Output: *selected_polytomies*: Dictionary containing taxons of all the leaves in a polytomy
<key: Node object from input tree, Value: List of taxons involving in a polytomy>

tree_x: The clone of input tree

selected_polytomies_D: The randomly selected taxons of size Degree
<key: Node object from Input tree involving the polytomy, Value: List of polytomy
of size Degree>

copy_translation: As the Tree object changes this dictionary contains the translation
keys from input tree to the clone tree
<key:Node objects input tree, value: Node objects cloned tree>

```
def select_polytomies(tree_,dictionary_,Degree):
    #Selected_polytomies contains all the leaves under the subtree. It is a 2d
        dictionary[i][j] where i represents the
    #orginal polytomy and j represents the head_node of the subtree. Input dictionary_
        will have multiple polytomies from
    #the original tree
```

```
tree_x=dendropy.Tree(tree_)
nds3 = [nd for nd in tree_.postorder_node_iter()]
nds4 = [nd for nd in tree_x.postorder_node_iter()]
copy_translation={}
for i, n in enumerate(nds3):
    copy_translation[n]=nds4[i]

selected_polytomies={}
for node in dictionary_.keys():
   selected_polytomies[node]={}
   for j in dictionary_[node]:
      if j.is_leaf():
         selected_polytomies[node][j]=[j]
      else:
         selected_polytomies[node][j]=[]
         new_tree=Tree(seed_node=j)
         for i in new_tree.postorder_node_iter():
            if i.is_leaf():
               selected_polytomies[node][j].append(i)

#selected_polymtomie_D contains randomly selected polytomies. If the subtree contains
    less than D leaves than we select all
#else we randomize our selection
selected_polytomies_D={}
for poly in selected_polytomies.keys():
   selected_polytomies_D[poly]=[]
   for head_ in selected_polytomies[poly].keys():
      if (len(selected_polytomies[poly][head_]))<=Degree:
         selected_polytomies_D[poly]=
             selected_polytomies_D[poly]+selected_polytomies[poly][head_]
      else:
         selected_polytomies_D[poly]=selected_polytomies_D[poly]+random.sample(selected_polytomies[pol

#pprint.pprint(selected_polytomies_D)
return selected_polytomies_D,tree_x,selected_polytomies,copy_translation
```

3.1.3 Creating .nex file

In this process the get sequence function gets the name of the location of the concerning replicate. Along with it we also give it the dictionary created in Select polytomy function in a loop based on the keys . This dictionary contains the :
`<key:Node object of single polytomy in a replicate, Value:List of Taxons selected with size Degree >` . This function then goes over all the '.fas' files in the replicate and retrieves the Sequence concerning the taxon stored in dictionary's value. It then creates a '.nex' file for each of the files in replicate folder. Finally test.py is invoked which created a combined '.nex' file for all the '.nex' files created in replicate folder.

```
def write_nexus():
```

```
   os.system('python ./Fat_sa/test.py')

def get_sequences(file__,dictionary_,m):
   x='0000'
   file_list=[]
   for i in range(1,int(m)+1):
      file=(x[0:4-len(str(i))]+str(i))
      #file_='output_'+file+'.nex'
      dictionary_input=SeqIO.to_dict(SeqIO.parse(file__+file+'.fas','fasta'))
      dictionary_output={}

      for j in dictionary_.keys():
         for k in dictionary_[j]:
            if str(k.taxon) !='None':
               tax=str(k.taxon).strip('"').strip("'")
               if tax not in dictionary_input.keys():
                  continue
               else:
                  dictionary_output[k]=dictionary_input[tax][:100]
      with open(file__+file+'.fasta', "w+") as handle:
            SeqIO.write(dictionary_output.values(),handle,'fasta')
      AlignIO.convert(file__+file+'.fasta', "fasta",'Fat_sa/'+file+'.nex',"nexus","DNA")

   write_nexus()


test.py

from Bio.Nexus import Nexus
import os
from Bio import AlignIO


x='0000'
os. chdir('Fat_sa/')
file_list=[]
for i in range(1,1001):
   file=(x[0:4-len(str(i))]+str(i))
   file=file+'.nex'
   file_list.append(file)

nexi = [(fname, Nexus.Nexus(fname)) for fname in file_list]
combined = Nexus.combine(nexi)
combined.write_nexus_data(filename=open(".//CS581_ASTRAL_SVD//COMBINED.nex", "w+"))
```

3.1.4 SVD quartets instruction

Here the combined.nex file created by function in section 1.5.1 is used to run SVDquatets and the output is stored as *out_svd.tre*

```
exe COMBINED.nex;svd showScores=no evalQuartets=all qformat=qmc replace=no;savetrees
    file='out_svd.tre' format='newick' replace='yes';quit;
```

3.1.5 Re-graft

In this function the input is the set of dictionaries that contain polytomies , copy translation and trees. Then this function reads the output tree created by SVDquartets. In this tree computed by SVDquartets, we iterate over each leaf and query the polytomy to find the subtree to which the leaf belongs. We then replace the leaf with the subtree. We are calling the result of this replacement a 'modified tree.' After going over all the leaves in SVDquartets's output, we delete duplicates for a subtree and keep only one instance of a subtree in the modified subtree. Finally 'modified tre' is then re-grafted to the Node in 'collapsed.tree' . This function is called in a loop with respect to the polytomies. Hence this will run for every polytomy created because of branch collapse.

```
def regraft(tree_,dictionary_,dictionary_1,copy_translation):
   tax = TaxonNamespace()
   #read the output tree of SVDquartets
   tree = Tree.get(file=open('out_svd.tre', 'r'),
                schema="newick",
                tree_offset=0,
                taxon_namespace=tax,
                suppress_edge_lengths=True,
                preserve_underscores=True)
   print(tree)
   taxon_to_remove=set()
   for node in tree.postorder_node_iter():
      tax_svd='581'
      if str(node.taxon)!='None':
         tax_svd=str(node.taxon).strip('"').strip("'")
      for key in dictionary_1.keys():
           for nodes in dictionary_1[key]:
              if str(nodes.taxon) !='None':
                 tax=str(nodes.taxon).strip('"').strip("'")
                 if tax_svd==tax:
                    new_tree=Tree(seed_node=key).clone()
                    node.taxon.label=None
                    if node.is_leaf():
                       if tax not in taxon_to_remove:
                          node.add_child(new_tree)
                          lis_=set([str(i.taxon).strip('"').strip("'") for i in
                              dictionary_1[key] if str(i.taxon) !='None'])
                          taxon_to_remove=taxon_to_remove.union(set(lis_))
                       else:
                          tree.prune_nodes([node])
```

```
   for node in dictionary_.keys():
      for nd in tree_.postorder_node_iter():
         if str(copy_translation[node])==str(nd):
            nd.add_child(tree)

   print(tree)
   print('-------------------------------------------------------------')
   print(tree_)
   #print(tree_.prune_subtree(list(dictionary_1.keys())))
   merge_dict=[]
   #print(copy_translation)
   for i in dictionary_1.keys():
      for k in dictionary_1[i]:
         merge_dict.append(copy_translation[k])

   tree_.prune_nodes(merge_dict)



   print(tree_)

   return tree_
```

3.1.6 Tree-Comparison

We borrowed this code from Molloy & Warnow [1] to compare the trees. This used to compare USA's tree with true species trees.
compare.py

```
import dendropy
def compareTreesFromPath(treePath1, treePath2):
    print("Comparing {} with {}".format(treePath1, treePath2))
    tax = dendropy.TaxonNamespace()
    tr1 = dendropy.Tree.get(path=treePath1,
                            schema='newick',
                            rooting='force-unrooted',
                            taxon_namespace=tax,
                            preserve_underscores=True)
    tr2 = dendropy.Tree.get(path=treePath2,
                            schema='newick',
                            rooting='force-unrooted',
                            taxon_namespace=tax,
                            preserve_underscores=True)
    tr1.collapse_basal_bifurcation(set_as_unrooted_tree=True)
    tr2.collapse_basal_bifurcation(set_as_unrooted_tree=True)
    return compareDendropyTrees(tr1, tr2)
    #print("RF distance on %d shared leaves: %d" % (nl, fp + fn))
def compareDendropyTrees(tr1, tr2):
    from dendropy.calculate.treecompare \
        import false_positives_and_negatives
```

```
    lb1 = set([l.taxon.label for l in tr1.leaf_nodes()])
    lb2 = set([l.taxon.label for l in tr2.leaf_nodes()])
    com = lb1.intersection(lb2)
    if com != lb1 or com != lb2:
        com = list(com)
        tns = dendropy.TaxonNamespace(com)
        tr1.retain_taxa_with_labels(com)
        tr1.migrate_taxon_namespace(tns)
        tr2.retain_taxa_with_labels(com)
        tr2.migrate_taxon_namespace(tns)
    com = list(com)
    tr1.update_bipartitions()
    tr2.update_bipartitions()
    nl = len(com)
    ei1 = len(tr1.internal_edges(exclude_seed_edge=True))
    ei2 = len(tr2.internal_edges(exclude_seed_edge=True))
    [fp, fn] = false_positives_and_negatives(tr1, tr2)
    rf = float(fp + fn) / (ei1 + ei2)
    return (nl, ei1, ei2, fp, fn, rf)
```

3.1.7 RAxML

This file is a utility file used to run RAxML 8.2.12 in all experiments found in this paper. RAxML is used in the alignment data for gene numbers in [50,100,250,500,100] to estimate the gene trees from the alignment. This will produce a gene tree for each alignment.
rax.py

```
 import os

location='path-to-directory//Astral//Data_//alignments_miss//25tax-1000gen-0bps-500K-1E-6-rand//'

location1='path-to-directory//Astral//Data_//gene_miss//25tax-1000gen-100bps-500K-1E-6-rand//'


folder_name=['25tax-1000gen-0bps-500K-1E-6-rand-miss']
sub_folder_name_6=['05','11','13','16','20']

li=['100','1000','500','250','50']
x='0000'

for k in li:
   for j in sub_folder_name_6:
      for i in range(1,int(k)+1):
         file=(x[0:4-len(str(i))]+str(i))
         input_=location+j+'//'+file+'.fasta'
         output_='rax_ml_'+k+'___100'+j+'_'+file+'.tre'
         os.system('RAxMLHPC -p 12345 -s "'+input_+'" -n "'+output_+'" -m GTRGAMMA')
```

3.1.8 MERGE

This is used to merge all the gene trees created by the rax.py in previous step. The merged portion of this file is used as an input for ASTRAL
merge.py

```
import glob



folder_name=['25tax-1000gen-0bps-500K-1E-6-rand-miss']
sub_folder_name_6=['05','11','13','16','20']


li=['1000','100','500','250','50']

for k in li:
   for j in sub_folder_name_6:
      read_files = glob.glob("RAxML_bestTree.rax_ml_"+k+"___100"+j+"_*.tre")




      with open("RAxML_"+j+"_"+k+".tre", "wb") as outfile:
          for f in read_files:
              with open(f, "rb") as infile:
                  outfile.write(infile.read())
```

3.1.9 ASTRAL

This file is a utility file used to run ASTRAL-III in all experiments found in this paper.
astral.py

```
import os

sub_folder_name_6=['05','11','13','16','20']


li=['100','100','500','250','50']
x='0000'


def astral(input_,output):
   os.system('java -jar astral.5.7.8.jar -i'+input_+' -o'+output)


import time

f=open('path-to-directory\\CS581_ASTRAL_SVD\\timer.txt','w')
start=time.time()
```

```
for k in li:
   for j in sub_folder_name_6:
      astral('RAxML_'+j+'_'+k+'.tre',
          'path-to-directory\\CS581_ASTRAL_SVD\\svd_output\\astral_6_'+j+'_'+k+'.tre')

end= time.time()

f.write('astral, '+str(end-start))

f.close()
```

3.1.10 SVDquartets

This file is a utility file used to run SVDquartets in all experiments found in this paper.
svd.py

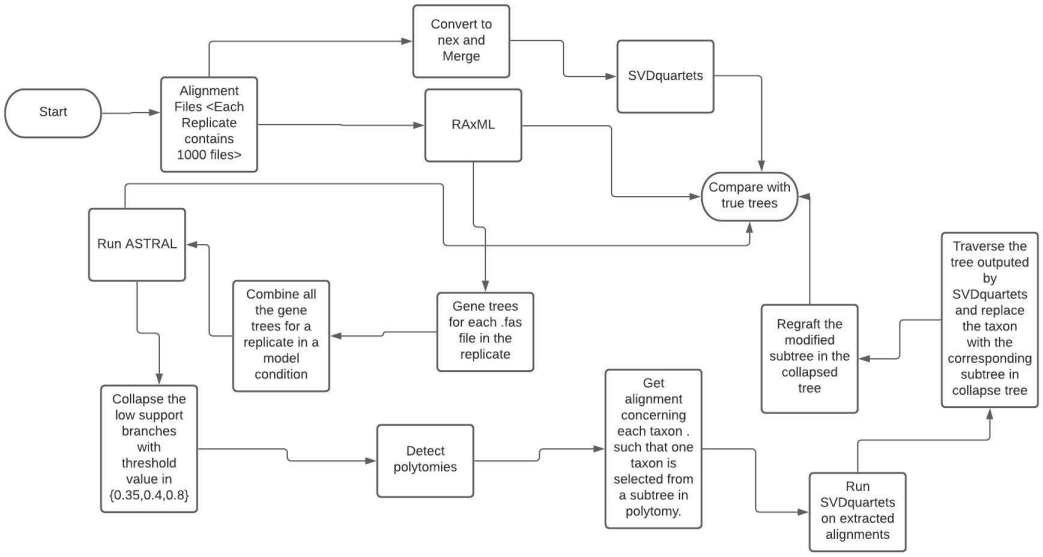
```
import os

sub_folder_name_6=['05','11','13','16','20']

li=['100','1000','500','250','50']

x='0000'


for k in li:
   for j in sub_folder_name_6:
      f=open('instrution_'+k+'_'+j+'.txt','w+')
      input_="COMBINED_"+k+"_"+j+".nex"
      print(input_)
      instrution= "exe COMBINED_"+k+"_"+j+".nex;svd showScores=no evalQuartets=all
          qformat=qmc replace=no;savetrees file='out_svd_"+k+"_"+j+".tre'format='newick'
          replace='yes';quit;"
      f.write(instrution)
      f.close()
```

```
import time

f=open('path-to-directory\\CS581_ASTRAL_SVD\\timer.txt','w')
start=time.time()
for k in li:
   for j in sub_folder_name_6:
      os.system('paup4c instrution_'+k+'_'+j+'.txt')

end= time.time()
```

```
f.write('SVDquartets, '+str(end-start))

f.close()
```

3.2 Experimental Workflow

The following chart shows how each method was analyzed in our experiments and results presented in this paper.



Experimental workflow of the studies done in this paper

3.3 Github

Finally, all scripts and programs as mentioned earlier can be found at the following Github: https://github.com/smishra677/CS581_ASTRAL_SVD. All information regarding the running of scripts can be found in the README file of the Github project.

3.3.1 Code pipeline

Pipeline: To run a python file open the terminal and run python3 ¡file_name¿.py

1. Data: The actual data can be found at https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main /input/25tax-1000gen-0bps-500K-1E-6-rand .The replicates used are 05, 11,13,16, and 20.

2. `svd_bps.py` There are 1000 alignment files(001-1000) in each replicate folder. These files are in .fas format. We then use the svd_bps.py, which can be found in https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main/Rax_ML svd_bps.py goes over all the 1000 fas files and selects the first 100 sites from the taxon in those files. It then

creates new alignment files with these 100 sites. The alignment files are named the same 001-1000 but have .fasta formatting. This file was primarily written by Sarthak.

3. `rax.py` After this, we run rax.py, which runs RAxML for all the alignment files created by svd_bps.py. rax.py and all the outputs created RAxML can be found in https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main/Rax_ML. This file was primarily written by Sarthak.

4. `merge.py` After running the RAxML we use merge.py to merge all the gene tree files created by rax.py. The merge happens for a replicate and the model condition. merge.py can be found in https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main/Rax_ML. This file was primarily written by Akhil.

5. `astral.py` Now we run astral using astral.py. This file can be found in https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main/Rax_ML This file was primarily written by Akhil.

6. `nex.py` Since we are done with astral we now use nex.py to convert all the fasta files created by nex.py to .nex file and to merge all the nex files concerning a replicate and the model condition. This file was primarily written by Sarthak.

7. `svd.py` We have created a combined nex file; we use svd.py, which first creates all the instructions.txt for the SVDquartets and runs it according to the instruction. We can find nex.py, svd.py https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main/svd. This file was primarily written by Sarthak.

8. `driver.py` Our method, USA, is implemented in driver.py, where it runs on three threshold sizes [0.35,0.4,0.8], and from each subtree, we are selecting one taxon. In terms of driver.py, both authors worked on this file although Sarthak took the lead and Akhil spent most of my effort refactoring code and tweaking due to feedback on preliminary results. Here Newick utility is called to collapse the branches with lower support values than the threshold, and then all the polytomies are extracted, and we run SVDquartets to resolve these polytomies. driver.py can be found at https://github.com/smishra677/CS581_ASTRAL_SVD

9. `display_results.py` The display_results.py script creates the graphical display of all results found in the svd_outputs/FinalData.csv file. Results can be grouped in comparing across methods and across threshold values, as well as comparing runtime results across methods as well. The display_results.py script can be found at https://github.com/smishra677/CS581_ASTRAL_SVD/blob/main/display_results.py. This file was primarily written by Akhil.

10. `comparison.py`: There is `comparision.py` which compares the output of our method with the true species tree. `comparision.py` uses dendropy to compare this and can be `comparision.py` at https://github.com/smishra677/CS581_ASTRAL_SVD/tree/main/svd_output. This file was primarily written by Akhil.

4 Labor and Work division

While both of us worked on each step of the project, the bolded name indicates who took the lead and spent more time working on each stage of the project.

Finalize project method	Akhil
Start running experiments	Sarthak
Finalize Experimental Conditions	**Akhil** and Sarthak
Create initial pipeline	Akhil and **Sarthak**
Run Preliminary Experiments	Akhil and **Sarthak**
Analyze results	**Akhil** and Sarthak
Refactor the code with Tandy's feedback	**Akhil** and Sarthak
Rerun the experiments with the refactored code	Akhil and **Sarthak**
Analyze the results and create graph	**Akhil** and Sarthak
Finish Report Write-Up	Akhil and Sarthak individually

Akhil worked on the initial algorithm construction and took lead on deciding the dataset model conditions to test as well as the experiment design to run. Akhil also focused on addressing some of the feedback given from the initial project progress reports and took the lead on writing code to address and fix some of those concerns. Finally, Akhil focused on the analysis of results and in visualizing the data generate from said experiments.

Sarthak worked on getting the initial dataset as well as running the preliminary results and saving the results. Sarthak focused on addressing some of the issues with incomplete data and in generating new data for some of the experiments. Sarthak also focused on running secondary experiments to address some of the concerns and issues addressed in feedback to the project progress reports.

Lastly, the final project write-up was written individually by each author albeit some information from the project proposal and project progress reports may seem similar between each person's write-up.

References

[1] Molloy, E. K. & Warnow, T. To Include or Not to Include: The Impact of Gene Filtering on Species Tree Estimation Methods. *Systematic Biology* **67**, 285–303 (2017). URL `https://doi.org/10.1093/sysbio/syx077`. `https://academic.oup.com/sysbio/article-pdf/67/2/285/24105597/syx077.pdf`.

[2] Zhang, C., Rabiee, M., Sayyari, E. & Mirarab, S. ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics* **19**, 153 (2018). URL `https://doi.org/10.1186/s12859-018-2129-y`.

[3] Stamatakis, A. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics* **22**, 2688–2690 (2006). URL `https://`

doi.org/10.1093/bioinformatics/btl446. https://academic.oup.com/bioinformatics/article-pdf/22/21/2688/16851699/btl446.pdf.

[4] Chou, J. *et al.* A comparative study of SVDquartets and other coalescent-based species tree estimation methods. *BMC Genomics* **16**, S2 (2015). URL https://doi.org/10.1186/1471-2164-16-S10-S2.

[5] Junier, T. & Zdobnov, E. M. The Newick utilities: high-throughput phylogenetic tree processing in the Unix shell. *Bioinformatics* **26**, 1669–1670 (2010). URL https://doi.org/10.1093/bioinformatics/btq243. https://academic.oup.com/bioinformatics/article-pdf/26/13/1669/512992/btq243.pdf.

[6] Sukumaran, J. & Holder, M. T. DendroPy: a Python library for phylogenetic computing. *Bioinformatics* **26**, 1569–1571 (2010).