

Growing phylogenetic trees with TreeLine

Erik S. Wright

November 1, 2022

Contents

1	Introduction	1
2	Performance Considerations	1
3	Growing a Phylogenetic Tree	2
4	Plotting Branch Support Values	5
5	Ancestral State Reconstruction	11
6	Exporting the Tree	13
7	Session Information	13

1 Introduction

This document describes how to grow phylogenetic trees using the `TreeLine` function in the DECIPHER package. `TreeLine` takes as input a set of aligned nucleotide or amino acid sequences and returns a phylogenetic tree (i.e., *dendrogram* object) as output. This vignette focuses on building maximum likelihood (ML) and maximum parsimony (MP) phylogenetic trees starting from sequences, but `TreeLine` can also be used to build additive trees from a distance matrix.

Why is the function called `TreeLine`? The goal of `TreeLine` is to find the most likely/parsimonious tree for a given sequence alignment. There are often many trees with nearly maximal likelihood/parsimony. Therefore, `TreeLine` seeks to find a tree as close as possible to the treeline, analogous to how no trees can grow above the treeline on a mountain.

Why use `TreeLine` versus other programs? The `TreeLine` function is designed to return an excellent phylogenetic tree with minimal user intervention. Many tree building programs have a large set of complex options for niche applications. In contrast, `TreeLine` simply builds a great tree when relying on its defaults. This vignette is intended to get you started and introduce additional options/functions that might be useful.

`TreeLine` uses multi-start optimization followed by hill-climbing to find the highest trees on the likelihood or parsimony landscapes. Since `TreeLine` is a stochastic optimizer, it optimizes many trees to prevent luck from influencing the final result. The main difference from most other approaches to tree optimization is that `TreeLine` heavily uses past trees to generate new trees as it optimizes. With any luck it'll find the treeline!

2 Performance Considerations

Finding a tree with very high likelihood/parsimony is no easy feat. `TreeLine` systematically optimizes hundreds to thousands of candidate trees before returning the best one. This takes time, but there are things you can do to make it

go faster.

- Only use the sequences you need: `TreeLine` scales a bit worse than quadratically with the number of sequences. Hence, limiting the number of sequences is a worthwhile consideration. In particular, always eliminate redundant sequences, as shown below, and remove any sequences that are not necessary. This concern is shared for all tree building programs, and `TreeLine` is no exception.
- Set a timeout: The `maxTime` argument specifies the (approximate) maximum number of hours you are willing to let `TreeLine` run. If you are concerned about the code running too long then simply specify this argument.
- Compile with OpenMP support: Significant speed-ups can be achieved with multi-threading using OpenMP. See the “Getting Started DECIPHERing” vignette for how to do this on your computer platform. Then you only need to set the argument `processors=NULL` and `TreeLine` will use all available processors.

3 Growing a Phylogenetic Tree

`TreeLine` takes as input a multiple sequence alignment when constructing a maximum likelihood or maximum parsimony phylogenetic tree. Multiple sequence alignments can be constructed from a set of (unaligned) sequences using `AlignSeqs` or related functions. `TreeLine` will optimize trees for amino acid (i.e., `AAStringSet`) or nucleotide (i.e., `DNAStrngSet` or `RNAStringSet`) sequences. Here, we are going to use a set of sequences that is included with DECIPHER. These sequences are from the internal transcribed spacer (ITS) between the 16S and 23S ribosomal RNA genes in several *Streptomyces* species.

```
> library(DECIPHER)
> # specify the path to your sequence file:
> fas <- "<<path to FASTA file>>"
> # OR find the example sequence file used in this tutorial:
> fas <- system.file("extdata", "Streptomyces_ITS_aligned.fas", package="DECIPHER")
> seqs <- readDNAStrngSet(fas) # use readAAStringSet for amino acid sequences
> seqs # the aligned sequences
DNAStrngSet object of length 88:
      width seq
[1] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC supercont3.1 of S...
[2] 627 NNNNCACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC supercont3.1 of S...
[3] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC supercont1.1 of S...
[4] 627 CGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC supercont1.1 of S...
[5] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC supercont1.1 of S...
...
[84] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC gi|297189896|ref|...
[85] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC gi|224581106|ref|...
[86] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC gi|224581106|ref|...
[87] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC gi|224581106|ref|...
[88] 627 TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC gi|224581108|ref|...
```

Many of these sequences are redundant or from the same genome. We can de-replicate the sequences to accelerate tree building:

```
> seqs <- unique(seqs) # remove duplicated sequences
> ns <- gsub("^.*Streptomyces( subsp\\. | sp\\. | | sp_) ([^ ]+).*$", "\\2", names(seqs))
> names(seqs) <- ns # name by species
> seqs <- seqs[!duplicated(ns)] # remove redundant sequences from the same species
> seqs
```

DNAStringSet object of length 19:

	width	seq	names
[1]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC	albus
[2]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC	clavuligerus
[3]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC	ghanaensis
[4]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC	griseoflavus
[5]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC	lividans
...	
[15]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC	cattleya
[16]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC	bingchenggensis
[17]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTTTCCGAATGGGGAAACC	avermitilis
[18]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC	C
[19]	627	TGTACACACCGCCCGTCA-CGTC...GGGGTGTCCGAATGGGGAAACC	Tu6071

Now, it's time to try our luck at finding the most likely tree. Here, we will set a stringent time limit (0.05 hours) to make this example faster, although longer time limits (e.g., 24 hours) are advised because setting very short time limits leaves the result up to luck.

Note that `TreeLine` automatically selects a substitution model based on Akaike information criterion (by default). It is possible to specify specific model(s) (e.g., `model="GTR+G4"`) to limit the possible selections and test your luck with fewer models.

Also, since `TreeLine` is a stochastic optimizer, it is critical to always set the random number seed for reproducibility. You can pick any lucky number, and if you every wonder how much you pushed your luck, you can try running again from a different random number seed to see how much the result came down to luck of the draw.

```
> set.seed(123) # set the random number seed
> tree <- TreeLine(seqs, reconstruct=TRUE, maxTime=0.05) # default is method="ML"
```

Optimizing model parameters:

```
JC69      -ln(L) = 5039, AICc = 10152, BIC = 10300
JC69+G4   -ln(L) = 4723, AICc = 9523, BIC = 9675
K80       -ln(L) = 4983, AICc = 10042, BIC = 10194
K80+G4    -ln(L) = 4413, AICc = 8905, BIC = 9061
F81       -ln(L) = 5020, AICc = 10121, BIC = 10281
F81+G4    -ln(L) = 4464, AICc = 9012, BIC = 9176
HKY85     -ln(L) = 4953, AICc = 9989, BIC = 10154
HKY85+G4  -ln(L) = 4371, AICc = 8829, BIC = 8997
T92       -ln(L) = 4965, AICc = 10010, BIC = 10166
T92+G4    -ln(L) = 4394, AICc = 8869, BIC = 9029
TN93      -ln(L) = 4942, AICc = 9969, BIC = 10138
TN93+G4   -ln(L) = 4371, AICc = 8831, BIC = 9004
SYM       -ln(L) = 4954, AICc = 9995, BIC = 10163
SYM+G4    -ln(L) = 4406, AICc = 8900, BIC = 9072
GTR       -ln(L) = 4929, AICc = 9951, BIC = 10132
GTR+G4    -ln(L) = 4368, AICc = 8832, BIC = 9017
```

The selected model was: HKY85+G4

PHASE 1 OF 3: INITIAL TREES

```
1/3. Optimizing initial tree #1 of 10 to 100:
-ln(L) = 4369.3 (-0.048%), 1 Climb
1/3. Optimizing initial tree #2 of 10 to 100:
-ln(L) = 4403.7 (+0.781%), 2 Climbs
1/3. Optimizing initial tree #3 of 11 to 100:
-ln(L) = 4367.8 (-0.033%), 6 Climbs
1/3. Optimizing initial tree #4 of 12 to 100:
-ln(L) = 4367.8 (+0.001%), 7 Climbs
```

PHASE 2 OF 3: REGROW GENERATION 1 OF 10 TO 20

```
2/3. Optimizing regrown tree #1 of 10 to 100:
-ln(L) = 4367.3 (-0.012%), 3 Climbs
2/3. Optimizing regrown tree #2 of 11 to 100:
-ln(L) = 4367.8 (+0.013%), 2 Climbs
2/3. Optimizing regrown tree #3 of 11 to 100:
-ln(L) = 4367.8 (+0.013%), 1 Climb
2/3. Optimizing regrown tree #4 of 11 to 100:
-ln(L) = 4367.4 (+0.003%), 1 Climb
2/3. Optimizing regrown tree #5 of 11 to 100:
-ln(L) = 4367.3 (~0.000%), 1 Climb
2/3. Optimizing regrown tree #6 of 11 to 100:
-ln(L) = 4367.8 (+0.013%), 0 Climbs
2/3. Optimizing regrown tree #7 of 11 to 100:
-ln(L) = 4369.3 (+0.045%), 1 Climb
2/3. Optimizing regrown tree #8 of 11 to 100:
-ln(L) = 4367.3 (~0.000%), 1 Climb
2/3. Optimizing regrown tree #9 of 11 to 100:
-ln(L) = 4367.3 (~0.000%), 3 Climbs
2/3. Optimizing regrown tree #10 of 11 to 100:
-ln(L) = 4367.3 (~0.000%), 3 Climbs
2/3. Optimizing regrown tree #11 of 11 to 100:
-ln(L) = 4367.7 (+0.009%), 1 Climb
```

4 Plotting Branch Support Values

Maybe it was just beginner's luck, but we already have a reasonable looking starting tree! TreeLine automatically returns a variety of information about the tree that can be accessed with the `attributes` and `attr` functions:

```
> attributes(tree) # view all attributes
$members
[1] 19

$height
[1] 2.366164

$state
[1] "-----CACCGCCCGTCA-CGTCACGAAAGTCGTAACACCCGAAGCCGGTGGCCCAACCCCCCG-GGGAGGGAGCCGTCGAA

$class
[1] "dendrogram"

$siteLnLs
 [1] -1.971719 -1.584704 -1.971719 -2.323353 -2.012399 -2.323353
 [7] -2.161851 -2.477202 -2.161851 -2.161851 -1.720633 -2.161851
[13] -2.161851 -2.161851 -1.720633 -2.120228 -2.161851 -2.477202
[19] 0.000000 -2.161851 -1.720633 -2.120228 -2.161851 -2.477202
[25] -4.662328 -1.720633 -2.477202 -2.477202 -2.477202 -1.720633
[31] -2.120228 -2.161851 -1.720633 -1.720633 -2.120228 -2.477202
[37] -5.914523 -4.662328 -5.914523 -2.161851 -2.161851 -2.161851
[43] -1.720633 -2.477202 -2.477202 -1.720633 -2.161851 -2.161851
[49] -4.949520 -4.454985 -2.120228 -1.720633 -1.720633 -2.161851
[55] -2.161851 -2.161851 -2.477202 -2.477202 -2.161851 -2.161851
[61] -2.161851 -12.604530 -7.361007 -16.118232 -13.964283 -6.790323
[67] -1.720633 -1.720633 -1.720633 -2.477202 -1.720633 -1.720633
[73] -1.720633 -2.477202 -1.720633 -8.779656 -11.923868 -1.720633
[79] -2.120228 -2.161851 -1.720633 -2.477202 -2.477202 -1.720633
[85] -1.720633 -2.120228 -1.720633 -1.720633 -1.720633 -2.477202
[91] -2.161851 -13.465148 -12.619613 -1.720633 -2.161851 -1.720633
[97] -2.477202 -2.120228 -2.120228 -1.720633 -1.720633 -1.720633
[103] -2.477202 -2.161851 -1.720633 -2.477202 -2.477202 -1.720633
[109] -2.120228 -2.161851 -1.720633 -2.120228 -2.477202 -2.477202
[115] -2.161851 -2.477202 -2.477202 -1.720633 -1.720633 -2.120228
[121] -2.477202 -1.720633 -2.161851 -2.161851 -1.720633 -2.120228
[127] -2.477202 -2.161851 -2.161851 -1.720633 -1.720633 -2.477202
[133] -2.477202 -1.720633 -1.720633 -2.120228 -1.720633 -2.161851
[139] -1.720633 -1.720633 -2.161851 -2.120228 -1.720633 -1.720633
[145] -2.477202 -2.120228 -2.161851 -2.477202 -2.161851 -2.161851
[151] -2.120228 -2.161851 -2.161851 -2.120228 -2.120228 -2.120228
[157] -2.161851 -2.120228 -2.477202 -2.477202 -1.720633 -1.720633
[163] -2.477202 -1.720633 -2.161851 -2.477202 -14.763598 -19.627395
[169] -5.536467 -12.081387 -12.345703 -13.656023 -17.145553 -13.344586
[175] -11.492012 -17.626131 -7.706616 -9.033263 -8.335405 -15.818433
[181] -22.587610 -19.605944 -7.738441 -12.275514 -11.205454 -10.982224
[187] -8.674655 -12.798941 -9.768623 -11.354638 -14.362479 -11.161633
```

[193]	-7.019592	-7.291542	-6.047315	-10.180352	-9.309122	-8.592309
[199]	-13.996526	-11.065195	-13.816095	-12.042061	-8.610045	-2.144336
[205]	-7.703378	-6.656128	-17.645138	-15.485550	-16.856899	-8.722252
[211]	-15.649818	-10.148589	-14.434057	-14.445693	-6.561292	-2.161851
[217]	-15.414665	-9.778316	-13.375509	-16.586161	-1.720633	-15.623164
[223]	-17.693751	-15.834543	-16.540740	-15.107939	-18.710164	-12.451975
[229]	-15.683075	-9.562813	-1.720633	-4.913289	-2.120228	-15.810547
[235]	-2.397450	-4.454985	-2.161851	-5.716830	-2.161851	-5.950753
[241]	-11.586018	-1.720633	-4.949520	-4.454985	-4.646789	-1.720633
[247]	-4.949520	-4.748700	-4.748700	-4.662328	-1.720633	-2.120228
[253]	-2.120228	-1.720633	-5.950753	-17.309594	-5.062351	-4.748700
[259]	-15.801553	-4.646789	-7.601980	-4.913289	-1.720633	-7.983113
[265]	-10.357135	-15.736423	-19.039724	-19.274449	-13.264776	-21.882792
[271]	-20.463036	-21.976439	-24.561516	-23.488739	-20.639679	-19.013786
[277]	-22.500606	-25.282615	-23.231611	-27.038538	-21.373324	-24.213062
[283]	-17.666024	-11.068328	-18.334900	-13.435904	-19.448785	-21.612242
[289]	-4.748700	-1.720633	-2.120228	-5.914523	-5.771228	-2.120228
[295]	-7.094834	-5.116749	-11.529632	-14.154008	-13.045383	-6.035188
[301]	-5.250470	-7.178567	-10.691212	-10.274557	-8.330944	-1.720633
[307]	-13.757732	-7.342707	-9.678997	-1.720633	-10.730702	-9.224347
[313]	-8.997023	-25.388677	-17.487960	-15.755714	-1.451778	-1.415547
[319]	-1.415547	-7.845294	-21.315832	-23.726932	-21.955225	-22.449983
[325]	-26.797052	-24.349447	-22.948484	-21.531984	-22.486037	-19.635422
[331]	-12.959099	-3.788214	-19.068484	-23.741485	-23.691021	-18.531589
[337]	-21.231652	-23.425754	-17.470855	-12.746396	-17.176219	-8.652325
[343]	-16.639278	-5.116749	-8.885816	-1.720633	-1.720633	-5.771228
[349]	-8.416308	-2.161851	-7.646144	-12.534144	-4.646789	-1.720633
[355]	-5.716830	-5.062351	-1.720633	-1.451778	-1.720633	-1.720633
[361]	-5.716830	-13.640510	-11.824091	-2.161851	-2.120228	-1.720633
[367]	-4.748700	-10.637787	-1.720633	-1.720633	-23.802642	-12.378439
[373]	-7.526115	-1.720633	-11.205761	-17.203148	-4.546690	-16.957118
[379]	-17.603541	-4.515675	-1.629043	-1.415547	-21.429827	-21.225615
[385]	-23.416460	-8.531079	-20.344703	-12.596573	-22.936429	-14.781788
[391]	-20.653578	-16.345640	-15.388213	-17.348296	-9.806466	-17.353529
[397]	-24.289752	-16.285216	-1.889180	-1.451778	-1.415547	-1.415547
[403]	-10.051695	-19.602496	-12.252111	-7.988164	-4.662328	-5.116749
[409]	-1.720633	-12.842073	-9.528564	-2.161851	-6.624487	-4.662328
[415]	-17.971640	-8.237922	-7.546549	-17.314211	-32.820522	-2.477202
[421]	0.000000	-9.454675	-9.309889	-14.233259	-21.281049	-28.822197
[427]	-21.141968	-19.982706	-21.717750	-17.359643	-18.050722	-17.749321
[433]	-21.411909	-30.206897	-14.999353	-24.935250	-23.518152	-23.855045
[439]	-22.527297	-14.021479	-19.613955	-15.798243	-1.720633	-7.413235
[445]	-8.494848	-15.910627	-18.963170	-4.913289	-1.720633	-1.720633
[451]	-7.413235	-1.720633	-5.640047	-15.793334	-6.663945	-1.584704
[457]	-1.584704	-1.971719	-4.662328	-1.720633	-5.062351	-2.120228
[463]	-1.720633	-13.005395	-2.120228	-2.120228	-1.720633	-2.477202
[469]	-1.720633	-2.477202	-2.477202	-2.161851	-9.660439	-10.668789
[475]	-4.662328	-2.477202	-12.000877	-2.477202	-1.720633	-2.120228
[481]	-1.720633	-1.720633	-2.477202	-4.662328	-1.720633	-2.161851
[487]	-8.719360	-2.477202	-1.720633	-2.161851	-2.477202	-2.120228
[493]	-2.161851	-2.120228	-1.415547	-1.415547	-1.720633	-2.120228

```

[499] -1.720633 -1.720633 -4.662328 -2.161851 -2.477202 -2.477202
[505] -1.720633 -2.120228 -2.120228 -5.062351 -2.120228 -2.120228
[511] -2.477202 -2.477202 -1.720633 -1.720633 -1.720633 -2.161851
[517] -4.454985 -2.161851 -2.477202 -4.662328 -1.720633 -1.720633
[523] -2.120228 -1.720633 -1.720633 -2.477202 -2.120228 -1.720633
[529] -4.662328 -2.161851 -2.120228 -2.120228 -1.720633 -1.720633
[535] -11.119173 -5.914523 -4.662328 -2.161851 -2.477202 -1.720633
[541] -1.720633 -2.477202 -4.748700 -2.161851 -2.161851 -1.720633
[547] -2.477202 -2.120228 -1.720633 -2.477202 -2.477202 -1.720633
[553] -1.720633 -2.477202 -2.161851 -1.720633 -2.120228 -1.720633
[559] -6.561292 -1.720633 -2.477202 -1.720633 -1.720633 -2.161851
[565] -4.662328 -10.445793 -2.161851 -1.720633 -2.477202 -2.120228
[571] -2.477202 -4.913289 -17.955495 -2.161851 -2.161851 -4.662328
[577] -2.161851 -1.720633 -1.720633 -1.720633 -1.720633 -2.477202
[583] -1.720633 -18.235352 -11.072067 -1.720633 -6.925109 -2.161851
[589] -2.477202 -2.477202 -2.161851 -6.957642 -10.445793 -11.603069
[595] -1.720633 -2.161851 -2.120228 -11.278956 -2.120228 -1.720633
[601] -2.477202 -2.120228 -2.161851 -2.161851 -1.720633 -4.454985
[607] -1.720633 -1.720633 -4.454985 -2.120228 -18.324415 -2.120228
[613] -2.161851 -2.161851 -1.720633 -2.477202 -2.477202 -2.120228
[619] -1.720633 -1.720633 -1.720633 -1.720633 -2.477202 -2.477202
[625] -2.477202 -2.161851 -2.161851

```

```

$method
[1] "ML"

```

```

$model
[1] "HKY85+G4"

```

```

$parameters
      FreqA      FreqC      FreqG      FreqT      FreqI      A/G      C/T      A/C
0.1788743 0.2341537 0.3441792      NA      NA 3.5915312      NA      NA
      A/T      C/G      Indels      alpha
      NA      NA      NA 0.1902499

```

```

$score
[1] 4365.7

```

```

$midpoint
[1] 10.64355

```

```

> attr("tree", "score") # best score
[1] 4365.7

```

The tree is (virtually) rooted at its midpoint by default. For maximum likelihood trees, all internal nodes include aBayes branch support values [1]. These are given as probabilities that can be used in plotting on top of each edge. We can also italicize the species names.

```

> plot(dendrapply(tree,
  function(x) {
    s <- attr(x, "probability") # choose "probability" (aBayes) or "support"
    if (!is.null(s) && !is.na(s)) {
      s <- formatC(as.numeric(s), digits=2, format="f")
      attr(x, "edgetext") <- paste(s, "\n")
    }
    attr(x, "edgePar") <- list(p.col=NA, p.lwd=1e-5, t.col="#CC55AA", t.cex=1.5)
    if (is.leaf(x))
      attr(x, "nodePar") <- list(lab.font=3, pch=NA)
    x
  }
),
  horiz=TRUE,
  yaxt='n')
> # add a scale bar
> arrows(0, 0, 0.4, 0, code=3, angle=90, len=0.05, xpd=TRUE)
> text(0.2, 0, "0.4 subs./site", pos=3, xpd=TRUE)

```



Figure 2: Tree with (aBayes) support probabilities at each internal node.

We lucked out because maximum likelihood and maximum parsimony trees both provide branch supports in the form of the fraction of optimized trees that contained a given partition (branch). These are accessible from the “support” attribute. As luck would have it, support values and (aBayes) probabilities are correlated, but support tends to be more conservative.

```

> getSupports <- function(x) {
  if (is.leaf(x)) {
    NULL
  } else {
    rbind(cbind(attr(x, "support"), attr(x, "probability")),
          getSupports(x[[1]]), getSupports(x[[2]]))
  }
}
> support <- getSupports(tree)
> plot(support[, 1], support[, 2], xlab="Support", ylab="aBayes probability", asp=1)
> abline(a=0, b=1, lty=2) # line of identity (y=x)

```

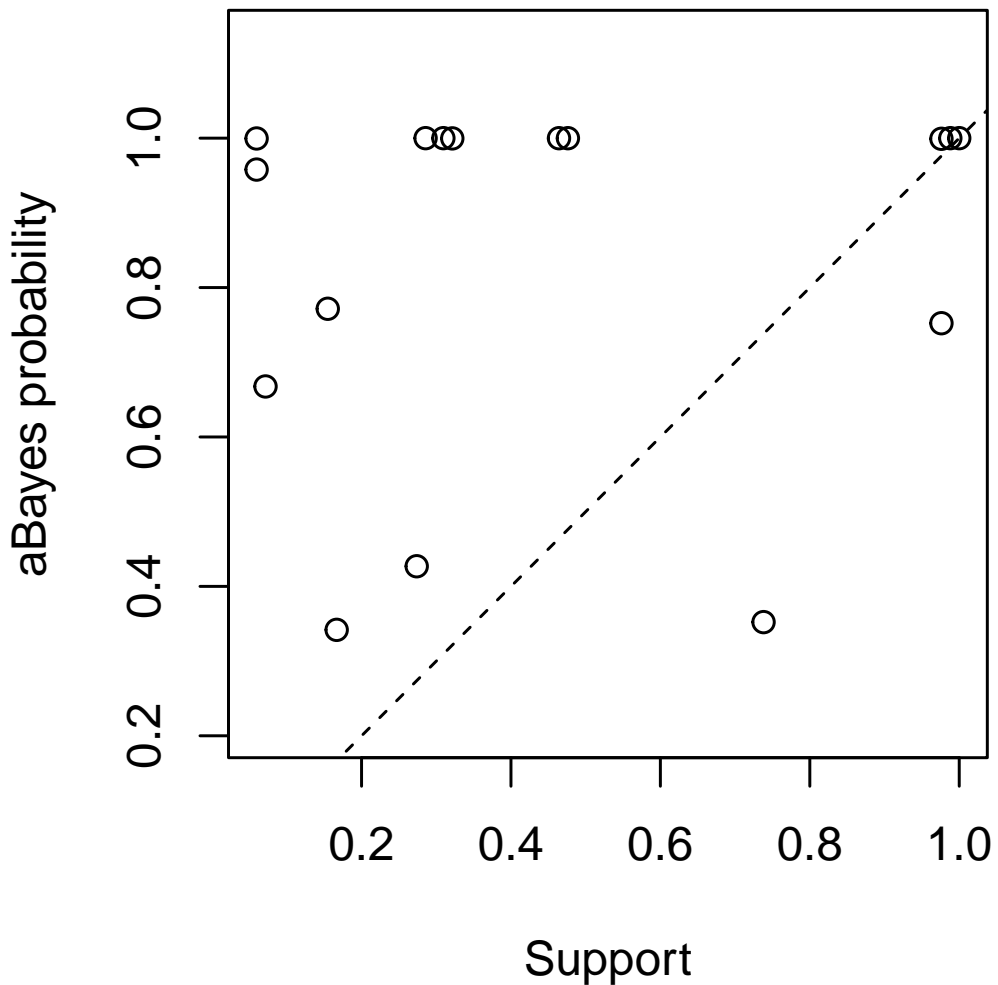


Figure 3: Comparison of aBayes probabilities and branch support values.

5 Ancestral State Reconstruction

We're in luck —one of the advantages of maximum likelihood and maximum parsimony tree building methods is that they automatically predict states at each internal node on the tree [2]. This feature is enabled when *reconstruct* is set to `TRUE`. These character states can be used by the function `MapCharacters` to determine state transitions along each edge of the tree.

```

> new_tree <- MapCharacters(tree, labelEdges=TRUE)
> plot(new_tree, edgePar=list(p.col=NA, p.lwd=1e-5, t.col="#55CC99", t.cex=0.7))
> attr(new_tree[[1]], "change") # state changes on first branch left of (virtual) root
[1] "G65T" "G168T" "G171T" "G172C" "G173A" "G180T" "G181C" "G182C" "G184T"
[10] "G185T" "G186A" "G201A" "G208T" "G211A" "G227T" "G259C" "G271T" "G272C"
[19] "G276T" "G277C" "G280A" "G282A" "G287C" "G288C" "G302A" "G303A" "G314C"
[28] "G321T" "G323A" "G324T" "G325C" "G326T" "G327T" "G328C" "G333C" "G337T"
[37] "G338T" "G339C" "G371C" "G379C" "G385A" "G389C" "G393T" "G396T" "G397C"
[46] "G419C" "G435A" "G437C" "G440T" "G447C" "G584C"

```

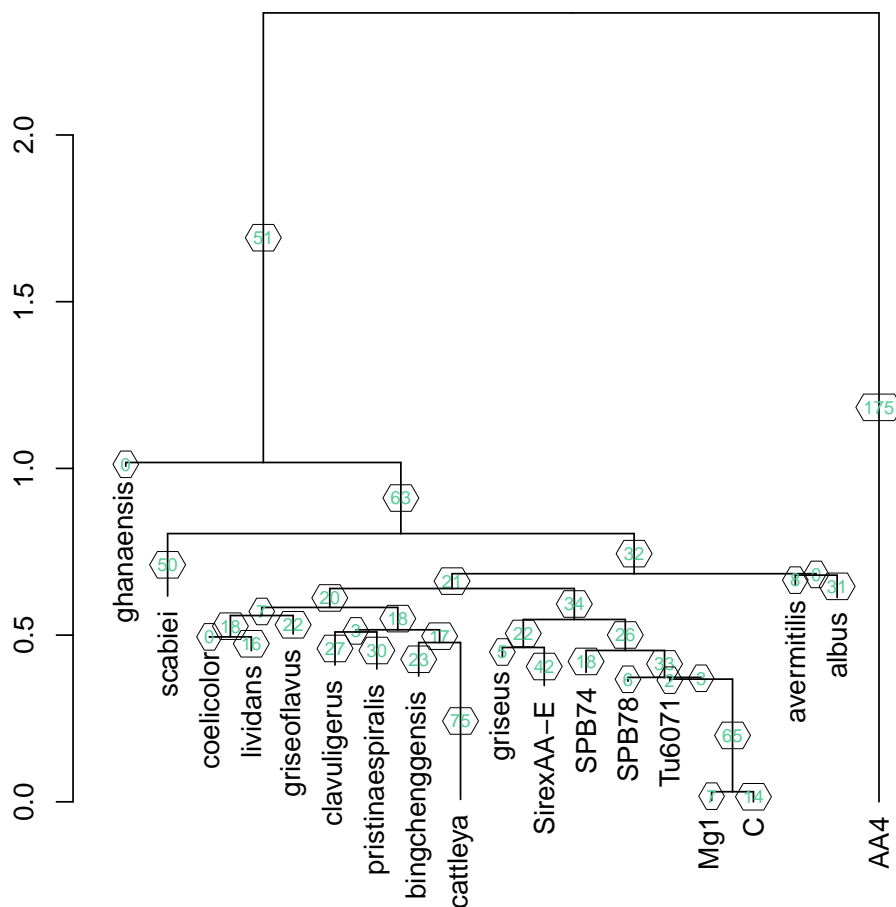


Figure 4: Edges labeled with the number of state transitions.

6 Exporting the Tree

We've had a run of good luck with this tree, so we'd better save it before our luck runs out! The functions `ReadDendrogram` and `WriteDendrogram` will import and export trees in Newick file format. If we leave the *file* argument blank then it will print the output to the console for our viewing:

```
> WriteDendrogram(tree, file="")
(("ghanaensis":0.01089221, ("scabiei":0.1873456, (((("coelicolor":2.661e-07, "lividans":0.
```

To keep up our lucky streak, we should probably include any model parameters in the output along with the tree. Luckily, Newick format supports square brackets (i.e., “[]”) for comments, which we can append to the end of the file for good luck:

```
> params <- attr(tree, "parameters")
> cat("[", paste(names(params), params, sep="=", collapse=","), "]", sep=" ", append=TRUE
[FreqA=0.178874322354543, FreqC=0.234153668368682, FreqG=0.344179204757228, FreqT=NA, FreqI=
```

7 Session Information

All of the output in this vignette was produced under the following conditions:

- R version 4.2.1 (2022-06-23), x86_64-pc-linux-gnu
- Running under: Ubuntu 20.04.5 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.16-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.16-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.44.0, Biostrings 2.66.0, DECIPHER 2.26.0, GenomeInfoDb 1.34.0, IRanges 2.32.0, RSQLite 2.2.18, S4Vectors 0.36.0, XVector 0.38.0
- Loaded via a namespace (and not attached): DBI 1.1.3, GenomeInfoDbData 1.2.9, KernSmooth 2.23-20, RCurl 1.98-1.9, Rcpp 1.0.9, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.3, cachem 1.0.6, cli 3.4.1, compiler 4.2.1, crayon 1.5.2, fastmap 1.1.0, memoise 2.0.1, pkgconfig 2.0.3, rlang 1.0.6, tools 4.2.1, vctrs 0.5.0, zlibbioc 1.44.0

References

- [1] Anisimova, M., Gil, M., Dufayard, J., Dessimoz, C., & Gascuel, O. Survey of branch support methods demonstrates accuracy, power, and robustness of fast likelihood-based approximation schemes. *Syst Biol.*, 60(5), 685-699.
- [2] Joy, J., Liang, R., McCloskey, R., Nguyen, T., & Poon, A. Ancestral Reconstruction. *PLoS Comp. Biol.*, 12(7), e1004763.