Evaluation of Branch Support Methods within the SVDQuartets Pipeline

Anton Goretsky[1], Yifan Yang[1], and Mollie Shichman,[1]

[1] *Department of Computer Science, University of Maryland, College Park*

Abstract

While very commonly used, traditional bootstrap methods may compound support for incorrect branches in phylogenetic estimation. We present an evaluation of a new resampling method, RAWR, within the SVDQuartets pipeline, evaluated on gene trees. We discuss where this new resampling method succeeds and falls short on such applications, and propose changes to the method, and further evaluations to perform on the SVDQuartets pipeline.

1 Introduction

Bootstrapping methods are used within phylogenetic analyses to perform resampling and estimate confidence intervals for a generated tree. Felsenstein proposed the standard bootstrap resampling (Felsenstein [1985]) which samples columns from a Multiple Sequence Alignment (MSA) uniformly, assuming an independent distribution. The problem with these methods is that branch support can be inflated and shift an algorithms' final tree away from the ground truth. For instance, incorrect branches tend to lend strong support to other incorrect branches. A recently proposed algorithm, RAWR (RAndom Walk Resampling), seems to have overcome these issues and outperformed traditional bootstrapping. The RAWR algorithm works by taking samples from random walks beginning at a given starting points through the MSA, stopping upon reaching a set sample size. The result is then unaligned and realigned on the sampled genes. A gene tree is calculated and can be used as a support tree for the original tree estimated off the MSA (Wang et al. [2021]).

While RAWR showed great promise for showing support for a given tree, it has not been tested in the application of the SVDQuartets pipeline. Unlike other phylogeny estimation algorithms that use summary statistics or more computationally intensive Monte Carlo methods, SVDQuartets is based on algebraic statistics, which allows for both scalability and higher accuracy. The SVDQuartets algorithm uses an agreement matrix generated from a MSA made by a quartet of taxa to estimate unrooted trees (Chifman and Kubatko [2014]).

Bootstrapping is used to determine support for each possible quartet tree generated and tested by SVDQuartets. While it can be a useful metric, bootstrapping is less reliable on trees with shorter branch lengths and with increased complexity. Our goal was to compare how RAWR and more traditional bootstrapping perform within the SVDQuartets pipeline using both simulated and biological data. Our hope was to find that RAWR perfoms better within this more specific application of support in order to prove its robustness and usefulness as a general use support method.

2 Methods

We are testing SVDquartets with RAWR resampling as compared to bootstrap resampling.

2.1 SVDquartets

SVDquartets is a method to estimate trees using singular value decomposition (SVD) (Chifman and Kubatko [2014]). Traditional phylogeny estimation methods may use all data at once with a Bayesian framework like BEST (Liu and Pearl [2007]) or *BEAST (Heled and Drummond [2009]), or use Monte Carlo approaches which are more computationally expensive. And in the case of species trees (which will be discussed in future work), from each gene tree independently, such as with Maximum Tree (Liu et al. [2010]). SVDquartets uses all data but does not rely on a Bayesian framework. In particular, the method infers tree split using SVD. For every possible 4 taxa split, the method constructs a matrix based on the data pattern of the 4 taxa. A valid split has rank ≤ 10, whereas invalid splits have rank > 10. By computing the SVD of each matrix, we can fetch the valid quartets and compute a final tree.

2.2 Bootstrap Resampling

In the pipeline suggested by Chifman and Kubatko [2014] to compute support, they first take resampling of the input MSA, and generate a tree using the SVDquartet method, then compute support and other statistics for the model tree. Bootstrapping is a method proposed by Felsenstein [1985] to estimate the output tree. When performing bootstrap resampling, sites of the MSA are randomly selected and concatenated to generate resampled sequences. Using this approach and under the assumption of Felsenstein [1985], branches in the true model tree would have high frequency in the trees generated from the resampled trees. We will use this method as our bootstrap baseline comparison in this study.

2.3 RAWR Resampling

RAWR, or RAndom Walk Resampling, takes a different approach to resample sequences (Wang et al. [2021]). Instead of taking sites randomly, the method starts from a random initial site, and traverses through the sequence. Each time it advances, the algorithm has a probability p to go the opposite direction. Here we follow the setup by Wang et al. [2021] and use $p = 0.1$. A main property of the method is that it preserves the relative position of the resampled sequence. Sites that are adjacent in the original MSA are still adjacent after resampling.

2.4 Our Pipeline

Our pipeline is defined in Algorithm 1. For RAWR resampling, we closely follow the process defined in the work of Wang et al. [2021] which we unalign the resampled sequence and realign before computing SVDquartets.

2.5 Data Simulations

Simulated data were an important part of the experiment because it provides us with a ground truth tree with which we can generate support metrics for our trees generated using RAWR and

Algorithm 1: General Pipeline

1 Generate MSA
2 Convert MSA into fasta format
3 Run RAWR resampling / bootstrap resampling
4 If RAWR was used, run MAFFT to align the resampled sequences
5 Convert aligned sampled sequences to NEXUS format
6 Run SVDquartets with PAUP* to generate support trees using aligned sample sequences

bootstrapping. We used SimPhy (Mallo et al. [2015]) to generate ground truth gene trees through a script written for testing Molloy and Warnow's NJMerge algorithm (Molloy and Warnow [2018]). From these gene trees, we then ran INDELible (Fletcher and Yang [2009]) to generate MSAs for each taxon. Each gene tree was generated with 15 taxa with 1 member per population over 3000 generations with a mutation rate of 1.0×10^{-7} in order to reduce compute time while still providing enough data for our resampling methods to be effective. Each MSA was generated with 10 introns, 10 exons, and 10 Ultra-Conserved Elements (UCEs) to give an even distribution of different gene types and to provide enough genes to have a meaningful experiment without using too much computational power.

In total, we created seven different MSAs for three different types of comparisons. Our base case gene trees were generated using ten million time units, and the MSAs for each gene each had 1000 nucleotides, with indels randomly generated from a Zipf's distribution where $\alpha = 5$ and the maximum length for an indel was capped at 50.

Our first comparison was between different Zipf's distributions values. By altering the distribution from which indels were randomly generated, we alter the amount and length of indels in the MSAs, which shape how different taxa are from each other. The length and amount of indels also manipulate how important it is to preserve horizontal relationships between the MSA columns, as the more indels occur and the longer they are, the more important their relationship to the larger gene is to determining support for a particular branch of the tree. With this in mind, we generated 3 additional MSAs, all with the exact same gene tree as the base case, but with altered Zipf distributions. The first MSA generated for comparison was altered by simply removing the limit on the maximum length of an indel. The second MSA generated set the maximum indel length to 5, and order of magnitude smaller than the base case's limit. Finally, for the third MSA we changed the alpha value from 5 to 2. This reduces the gradient of the distribution, making it more likely that larger values of indels will be picked. However, we still capped the maximum length to 50 in order to have a clear cause and effect shown with our base case.

The second comparison was between the length of the generation. In theory, longer generations mean it is easier to distinguish the time point where a branch split. We thus generated a gene tree with the same number of generations and taxa, but with only 500,000 time units instead of 10 million. We chose that number from the script options given in Molloy's for running SimPhy (Molloy and Warnow [2018]).

Our final comparison was between the number of nucleotides per gene. This was to test if longer sequences helped increase support due to having more columns to randomly select from a MSA. We generated two sets of MSAs identical to the base case but with each gene having 1500 and 500 nucleotides, respectively.

2.6 Empirical Data

For empirical data, we used datasets from the CRW database (Robin Gutell) — the Comparative Rna Web Site and Project by the Gutell Lab from The University of Texas at Austin. Specifically, we used three intron RNA alignments groups: Group IA, IC2 and ID introns. Group IA has 110 taxa, length after alignment is 6166 characters. Group IC has 32 taxa, length after alignment is 3588 characters. Group ID has 25 taxa, length after alignment is 3114 characters. For larger datasets, the estimated runtime exceeds the capacity of this project, thus we stick to these three groups.

3 Results

In order to understand the performance of using RAWR in place of Bootstrapping within the SVDQuartets pipeline, we calculated the area under the ROC curve, and the area under the precision recall curve for each empirical and simulated dataset. Table 1 shows the areas for all empirical datasets used. Table 2 shows the averaged areas for each of the 30 generated sequences for each simulated data variation. Tables were also created for each independent generated sequence, those tables are available in the appendix. Blank cells indicate that there were no informative quartets for said sequence and/or method, and no tree was created by SVDQuartets. True positives (TP) are those bipartitions whose support is greater than or equal to a threshold, and are present in the reference tree. False positives (FP) are those bipartitions whose support is greater than or equal to a threshold, and are *not* present in the reference tree. True negatives (TN) are those bipartitions whose support is less than a threshold, and are present in the reference tree. False negatives (FN) are those bipartitions whose support is less than a threshold, and are not present in the reference tree. Precision is defined as the following, $\frac{TP}{TP+FP}$ and recall is defined as $\frac{TP}{TP+FN}$. The ROC, or receiver operating characteristic, is defined as the TP rate plotted against the FP rate.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
IGIA	0.37992	0.37992	0.19548	0.19548
IGIC2	0.52867	0.5827	0.33514	0.35053
IGID	0.60712	0.70933	0.40071	0.53921

Table 1: Empirical Data AUC-ROC and AUC-PR

Data	Average RAWR-AUC_ROC	Average Bootstrap-AUC_ROC	Average RAWR-AUC_PR	Average Bootstrap-AUC_PR
base_expr	0.7941	0.81735	0.66957	0.78194
gen_len_500K	0.21675	0.36888	0.35196	0.23826
gene_len_500	0.68208	0.77308	0.51842	0.6863
gene_len_1500	0.83429	0.84808	0.67185	0.81368
indel_lim_5	0.78284	0.83361	0.66171	0.78508
indel_no_lim	0.78621	0.82409	0.63453	0.77059
indel_pow_2	0.78264	0.81465	0.64247	0.77196

Table 2: Simulated Data Averaged AUC-ROC and AUC-PR

We can see for the empirical dataset IGIA, RAWR and Bootstrapping performed equally poor. We start to see the differences between the performance of RAWR and bootstrapping when we move on to IGIC2 and IGID in the empirical data. Here we see that traditional bootstrapping outperforms RAWR in both the AUC_POC and AUC_PR scores. In the simulated data, the same trend continues, traditional bootstrapping consistently outperforms RAWR on almost every simulated dataset, for both metrics. The only instance in which RAWR outperforms bootstrapping is on the gen_len_500K dataset under the AUC_PR metric. Of all the simulated datasets, this one also had the poorest performance overall on both bootstrapping and RAWR.

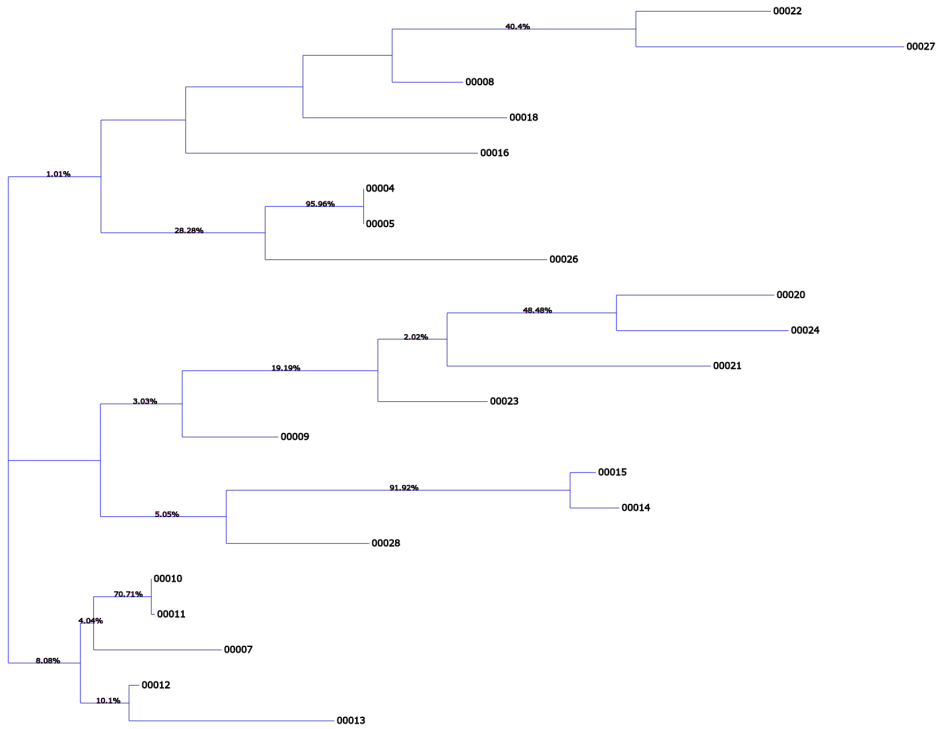

Figure 1: *Reference tree of* ***group ID introns*** *with support values annotated with* ***RAWR*** *resampled trees.*

4 Discussion

We see that scores for RAWR resampling are much lower in the empirical CRW dataset when compared to simulation datasets. RAWR's main assumption and property is that sites that are close in the original sequence are still close after resampling; in other words, they maintain their relative position, unlike in bootstrap resampling and most other resamplings (Wang et al. [2021]). However this property is not preserved if the original sequence is sparse. As the unalign process collapses all gaps, sites that are not adjacent in the MSA become adjacent. More importantly, when sampling in a sparse sequence with random walks, the pointer is easily stuck on a region with many gaps, outputing a near-empty sequence. This is why RAWR performs better on simulation data than empirical data, as simulated data is very dense.

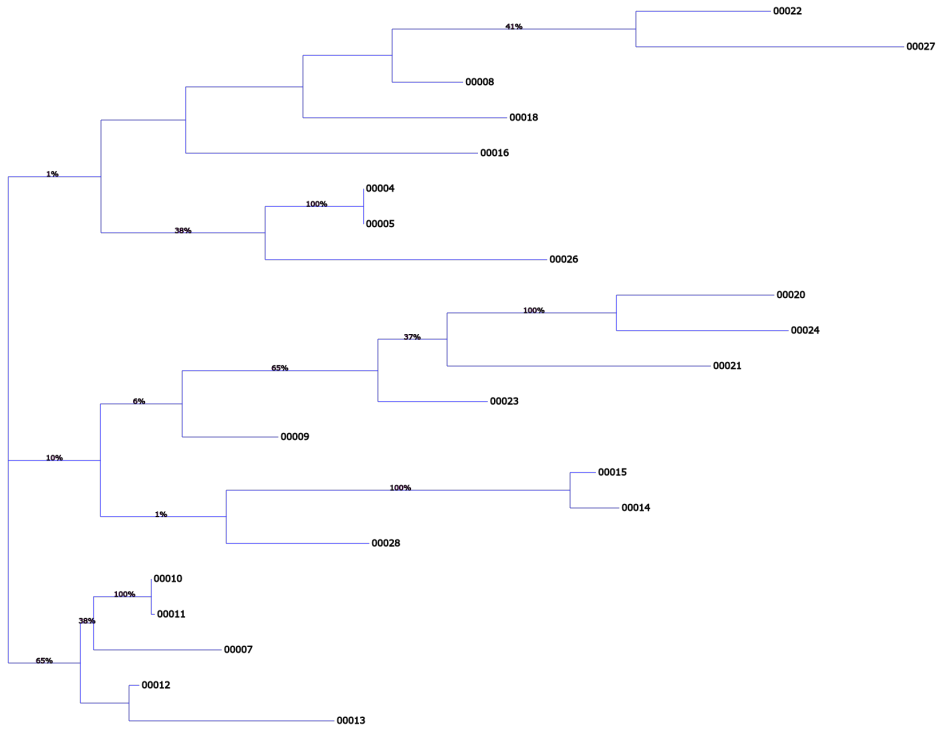

Figure 2: *Reference tree of* ***group ID introns*** *with support values annotated with* ***bootstrap*** *resampled trees.*

SVDquartets also fails for many RAWR resamplings. As showcased in table 4 and other simulation tables, resamples from RAWR can sometimes destroy the hidden tree relationship. In the work of Wang et al. [2021], they argue that bootstrap resampling does not need re-estimation of the MSA as the resampled sequence does not preserve sequence homology, where as RAWR can re-estimate the homology to correct and "sychronize" incorrect subsequences. For SVDquartets, as matrix decomposition is sensitive to disturbance, realigning the sequences can result in very different trees, making the ROC score inconsistent when applied to different datasets.

We can also see a significant difference in performance when it comes to sequence length. Dataset gene_len_1500 achieves higher scores under both resampling methods as compared to gene_len_500, which can also be attributed to the sensitivity of matrix decomposition. We don't see a significant difference between when varying the indel length limits and alpha values. Bootstrapping was helped slightly by a lower limit on the indel length, but not substantially enough that randomness couldn't explain it.

5 Conclusion

In this work, we test RAWR resampling against bootstrap resampling within the SVDquartets pipeline. In both the empirical and simulated data, we see that RAWR generally performs worse than bootstrap resampling. SVDquartets also fails more when using RAWR than when usign bootstrap. Rawr resampling works better when there's less time between generations, resulting in

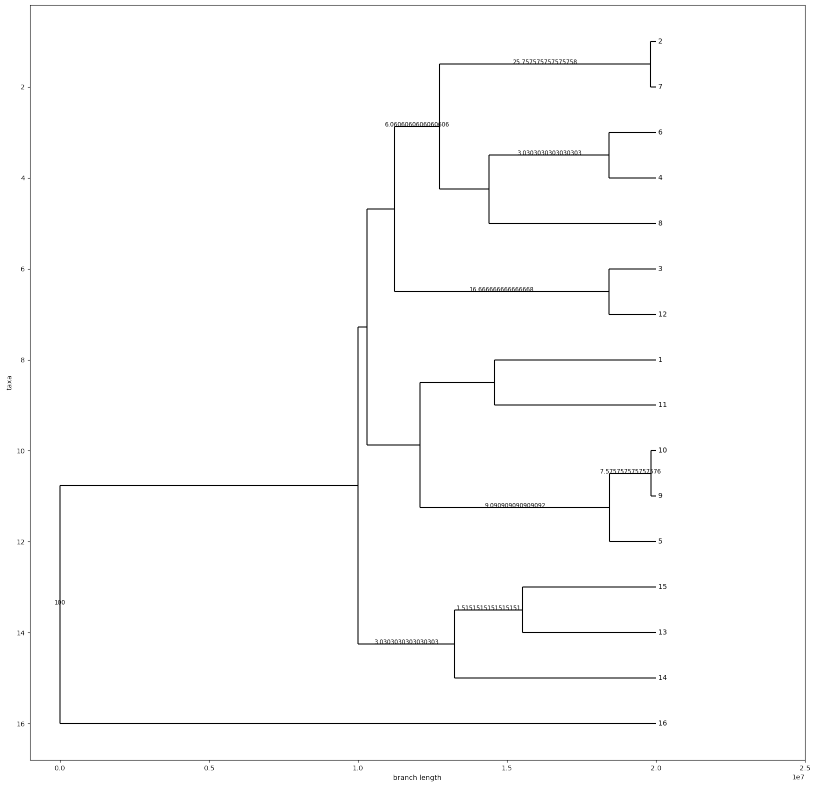

Figure 3: *Reference tree of* ***gene_len_500*** *with support values annotated with* ***RAWR*** *resampled trees.*

shorter branch length, whereas bootstrap works better when branches are longer.

6 Future Work

There are several directions for future work on this pipeline analysis. As we believe a large part of RAWR's poor performance on empirical data is due to sparsity, we would like to modify RAWR to largely walk through regions that have a lower amount of gaps, as this would allow it to sample already adjacent regions, preserving the adjacency property after unalignment.

More detailed analyses would also be of interest to perform. Those include testing a different variety of branch lengths, the gap percentage in MSA alignments, and the inclusion/exclusion of MAFFT in the RAWR resampling pipeline, to account for more of the variablility that may exist. We would also perform multiple trials of the experiment to account for the randomness in the simulated data, and report statistically significant testing results.

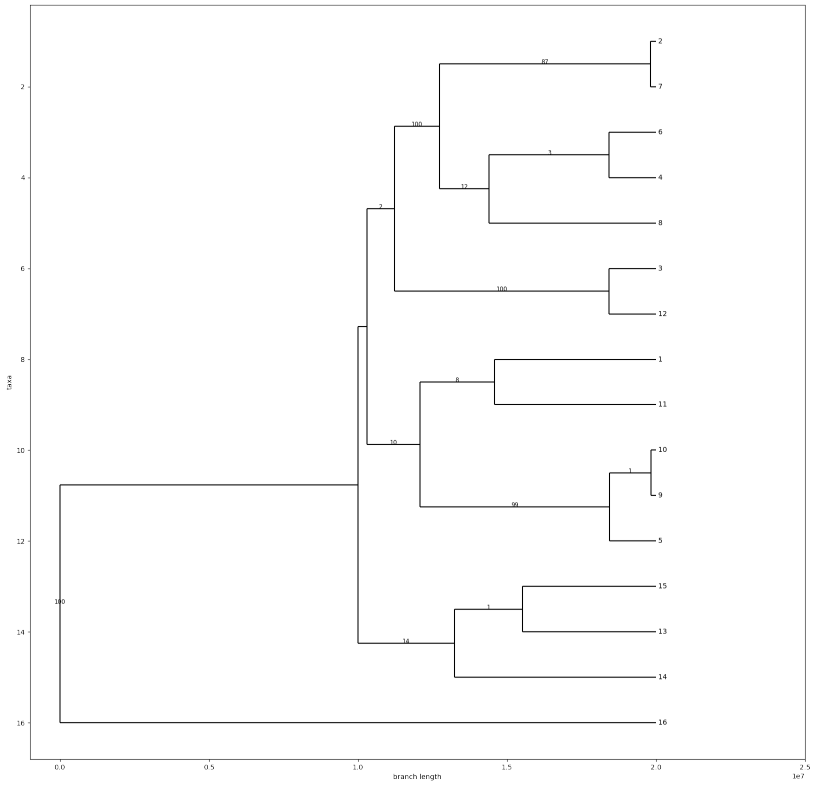

Figure 4: *Reference tree of* ***gene_len_500*** *with support values annotated with* ***bootstrap*** *resampled trees.*

Furthermore, this experiment was only run on gene trees. Ultimately, the SVDQuartets pipeline was originally created with the goal of generating and feeding highly accurate trees into supertree amalgamation algorithms like Quartet MaxCut to better determine a larger and more complete tree of life (Snir and Rao [2012]), aka for species tree estimation. There also exist more recent iterations of traditional bootstrapping, such as multi-locus bootstrapping, which does multiple rounds of sampling to account for the combination of alleles at different loci (Seo [2008]). It would prudent to extend this research to look at species trees, as was the original intent of the SVDQuartet pipeline.

7 Supplementary Materials

7.1 Simulations

We downloaded the uncompiled code for both SimPhy and INDELible. After downloading its dependencies using the brew command line tool, we compiled Simphy by running "make" in the main folder pulled from Github. We then moved the binary from SimPhy/bin to our own software folder. We then altered the variable SIMPHY from Malloy's script

`b_run_simphy.sh`

to reflect the new path to the SimPhy binary. We then could run the script by entering its directory and running ./b_run_simphy.sh. Within the script, we set ntax=15 and trln=500000 or 10000000.

Once we downloaded the material provided for INDELible, we entered its src directory and ran "g++ -o indelible -O4 indelible.cpp" to generate an executable (this command is found in src/readme.md). We then moved the executable from the downloaded INDELible source code folder to our tools folder, which contained the python scripts from Malloy and Warnow that we modified for our experiment. Specifically, we added two lines to the section of run_indelible.py that write the control.txt file that INDELible needs for its basic model parameters to include

```
f.write("[indelmodel] POW 5 50\n")
f.write("[indelrate] 0.0000001\n")
```

so that INDELible would generate MSAs with indels. The [indelmodel] line was what we modified for changing the distribution for random indel generation. Additionally, the script originally only used the os library in python to run bash commands, but for unknown reasons the script wouldn't complete after running the os command to run INDELible. We thus updated run_indelible.py to use python's subprocess library instead, which is the updated version of os. However, we still used the os library for changing directories and renaming and removing files, while subprocess removed directories, copied files, and ran INDELible. The command we used to generate the base MSA was

```
python3 run_indelible.py
-x ../../../tools/indelible
-s 0 -e 3000
-t ../data/15tax-3000gen-10M/01/true-genes.tre
-p indelparam10genes_1000
-o ../data/indelible_data
```

We would then modify the paths for -t to change the gene tree and -p to change the parameters folder. To generate the parameters file that gave the probabilities of each nucleotide pair occurring

and the length of each gene, we used the python script set_indelible_params.py given by Malloy and Warnow (Molloy and Warnow [2018]). We modified its build_len_vector function, which was originally used to generate genes of variable lengths, to return one length only, as follows

```
def build_length_vector(size, low=300, high=1500):
    seqlen = numpy.full(size, 1500)
    return seqlen
```

where we changed the second argument of numpy.full based on how many nucleotides per gene were required for our experiment. To generate the parameters file with set_indelible_params.py we ran the following in a mac terminal:

```
set_indelible_params.py -e 10 -i 10 -u 10 -o indelparam10genes_1500
```

where -e -i and -u set the number of exons, introns, and UCEs.

7.2 MAFFT

We used *MAFFT v7.490* to align sequences. The command we used to generate alignments was:

```
call mafft.bat --out target.aln.fasta --localpair input.seq.fasta
```

7.3 Format Conversion

We use *Biopython 1.79*, which is maintained by Kushinauwu. Each conversion can be used as the following:

```
python fasta2nexus.py -f input.fasta -n output.nex -m SEQUENCETYPE\\
python phylip2fasta.py -p input.phy -f target.fasta -m SEQUENCETYPE
```

All scripts are available at project folder.

7.4 Statistic Calculation

We modify the script by Wang et al. [2021] to calculate percision, recall, etc for support trees. The script can be used as following:

```
python CalAUC.py -r reference.tree -s support.trees -p PREFIX
```

Here support trees should be one file where each line is a support tree. Prefix is a string that contains output prefix and path. This script is available at project folder.

References

Julia Chifman and Laura Kubatko. Quartet inference from snp data under the coalescent model. *Bioinformatics*, 30(23):3317–3324, 2014.

Joseph Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *evolution*, 39(4):783–791, 1985.

William Fletcher and Ziheng Yang. INDELible: A Flexible Simulator of Biological Sequence Evolution. *Molecular Biology and Evolution*, 26(8):1879–1888, 05 2009. ISSN 0737-4038. doi: 10.1093/molbev/msp098. URL https://doi.org/10.1093/molbev/msp098.

Joseph Heled and Alexei J Drummond. Bayesian inference of species trees from multilocus data. *Molecular biology and evolution*, 27(3):570–580, 2009.

Kushinauwu. About the obf. URL https://www.open-bio.org/.

Liang Liu and Dennis K Pearl. Species trees from gene trees: reconstructing bayesian posterior distributions of a species phylogeny using estimated gene tree distributions. *Systematic biology*, 56(3):504–514, 2007.

Liang Liu, Lili Yu, and Dennis K Pearl. Maximum tree: a consistent estimator of the species tree. *Journal of mathematical biology*, 60(1):95–106, 2010.

Diego Mallo, Leonardo De Oliveira Martins, and David Posada. SimPhy : Phylogenomic Simulation of Gene, Locus, and Species Trees . *Systematic Biology*, 65(2):334–344, 11 2015. ISSN 1063-5157. doi: 10.1093/sysbio/syv082. URL https://doi.org/10.1093/sysbio/syv082.

Erin K. Molloy and Tandy Warnow. Data from: Statistically consistent divide-and-conquer pipelines for phylogeny estimation using njmerge, 2018. URL https://doi.org/10.13012/B2IDB-0569467_V1.

Jamie Cannone Robin Gutell. Comparative rna web site and project the gutell lab. URL https://crw-site.chemistry.gatech.edu/.

Tae-Kun Seo. Calculating bootstrap probabilities of phylogeny using multilocus sequence data. *Molecular Biology and Evolution*, 25(5):960–971, 2008. ISSN 18281270. doi: https://doi.org/10.1093/molbev/msn043.

Sagi Snir and Satish Rao. Quartet maxcut: A fast algorithm for amalgamating quartet trees. *Molecular Phylogenetics and Evolution*, 62(1):1–8, 2012. ISSN 1055-7903. doi: https://doi.org/10.1016/j.ympev.2011.06.021. URL https://www.sciencedirect.com/science/article/pii/S1055790311003101.

Wei Wang, Ahmad Hejasebazzi, Julia Zheng, and Kevin J Liu. Build a better bootstrap and the rawr shall beat a random path to your door: phylogenetic support estimation revisited. *Bioinformatics*, 37(Supplement_1):i111–i119, 2021.

8 Appendix

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
base_expr_1	0.74133	0.82505	0.49601	0.72624
base_expr_2	0.616	0.67548	0.40163	0.67443
base_expr_3	0.77703	0.83995	0.64834	0.83797
base_expr_4	0.78548	0.65119	0.56386	0.5339
base_expr_5	0.19231	0.90805	0.54167	0.618
base_expr_6	0.9155	0.90051	0.78489	0.85703
base_expr_7	0.81776	0.81742	0.673	0.71147
base_expr_8	0.82885	0.79704	0.58104	0.79123
base_expr_9	0.80648	0.83654	0.70977	0.82617
base_expr_10	0.68029	0.7758	0.46191	0.72321
base_expr_11	0.61535	0.7758	0.50106	0.72321
base_expr_12	0.91267	0.6453	0.88033	0.63305
base_expr_13	0.80999	0.90438	0.72151	0.90862
base_expr_14	0.78517	0.7967	0.54106	0.82089
base_expr_15	0.88386	0.72104	0.70522	0.67816
base_expr_16	0.90513	0.87019	0.82429	0.78206
base_expr_17	0.77267	0.82937	0.52383	0.8177
base_expr_18	0.90509	0.7875	0.82212	0.75957
base_expr_19	0.9227	0.8544	0.90434	0.89336
base_expr_20	0.91878	0.9308	0.84532	0.96182
base_expr_21	0.64903	0.92456	0.46804	0.94837
base_expr_22	0.92284	0.63474	0.87061	0.61529
base_expr_23	0.70356	0.87557	0.52514	0.89619
base_expr_24	0.9289	0.8	0.93654	0.78462
base_expr_25	0.90306	0.92521	0.65704	0.95512
base_expr_26	0.8463	0.86842	0.69044	0.80769
base_expr_27	0.74759	0.7517	0.60839	0.75232
base_expr_28	0.82143	0.8881	0.68653	0.8003
base_expr_29	0.91379	0.8188	0.8437	0.71105
base_expr_30		0.89098		0.90908

Table 3: AUC ROC and AUC for precision-recall curve for simulated data *base_expr*.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
gen_len_500K_1		0.33442		0.26327
gen_len_500K_2	0.10877	0.63581	0.21894	0.26473
gen_len_500K_3		0.14615		0.3449
gen_len_500K_4	0.27381	0.56056	0.28862	0.43381
gen_len_500K_5		0.23904		0.10403
gen_len_500K_6				
gen_len_500K_7		0.34172		0.2871
gen_len_500K_8		0.41818		0.10851
gen_len_500K_9	0.15247	0.35868	0.20801	0.22056
gen_len_500K_10	0.05	0.23132	0.28261	0.11703
gen_len_500K_11	0.13846	0.23132	0.47179	0.11703
gen_len_500K_12	0.25	0.23001	0.43333	0.19844
gen_len_500K_13	0.25	0.44527	0.43333	0.27755
gen_len_500K_14	0.45214	0.41949	0.28	0.21587
gen_len_500K_15		0.50933		0.36354
gen_len_500K_16	0.18029	0.35093	0.39328	0.11914
gen_len_500K_17	0.34156	0.19814	0.30112	0.13065
gen_len_500K_18		0.37579		0.21758
gen_len_500K_19	0.2967		0.51512	
gen_len_500K_20	0.31973	0.59087	0.23267	0.43326
gen_len_500K_21	0.15035	0.38644	0.33792	0.27977
gen_len_500K_22	0.16667	0.48109	0.40625	0.31998
gen_len_500K_23	0.17033	0.31669	0.41667	0.19376
gen_len_500K_24		0.40955		0.22304
gen_len_500K_25		0.2734		0.24649
gen_len_500K_26				
gen_len_500K_27				
gen_len_500K_28				
gen_len_500K_29				
gen_len_500K_30	0.16667		0.41176	

Table 4: AUC ROC and AUC for precision-recall curve for simulated data *gen_len_500K*.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
gene_len_500_1	0.53224	0.7914	0.40404	0.62421
gene_len_500_2	0.48989	0.51073	0.2907	0.40401
gene_len_500_3	0.49038	0.80817	0.2954	0.65332
gene_len_500_4	0.76376	0.73948	0.51087	0.55621
gene_len_500_5	0.80165	0.87004	0.5509	0.82213
gene_len_500_6	0.67214	0.76979	0.49672	0.69407
gene_len_500_7	0.8481	0.88626	0.64615	0.81029
gene_len_500_8	0.48214	0.8353	0.34384	0.64557
gene_len_500_9	0.6759	0.76219	0.51672	0.55368
gene_len_500_10	0.7219	0.65446	0.61658	0.71704
gene_len_500_11	0.50371	0.65446	0.43021	0.71704
gene_len_500_12	0.91677	0.47146	0.88464	0.53527
gene_len_500_13	0.80303	0.92426	0.66979	0.92259
gene_len_500_14	0.66615	0.69573	0.42372	0.63212
gene_len_500_15	0.8675	0.69545	0.62072	0.62085
gene_len_500_16	0.88133	0.89137	0.81135	0.76676
gene_len_500_17	0.40604	0.90293	0.25831	0.89972
gene_len_500_18	0.89431	0.77488	0.79772	0.61206
gene_len_500_19	0.83187	0.92143	0.74504	0.92477
gene_len_500_20	0.84816	0.88839	0.57811	0.88975
gene_len_500_21	0.62266	0.92147	0.42917	0.92311
gene_len_500_22	0.4921	0.66204	0.40602	0.59926
gene_len_500_23	0.43393	0.78761	0.46381	0.63188
gene_len_500_24	0.21978	0.62033	0.12092	0.58247
gene_len_500_25	0.66313	0.6869	0.41678	0.36736
gene_len_500_26	0.83401	0.79303	0.68847	0.49926
gene_len_500_27	0.82167	0.81303	0.61285	0.76864
gene_len_500_28	0.80795	0.82	0.56434	0.81697
gene_len_500_29	0.87898	0.76244	0.67306	0.58944
gene_len_500_30	0.59125	0.87722	0.28578	0.8091

Table 5: AUC ROC and AUC for precision-recall curve for simulated data $gene\_len_500$.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
gene_len_1500_1	0.89058	0.775	0.74016	0.79594
gene_len_1500_2	0.74286	0.83929	0.47725	0.84386
gene_len_1500_3	0.87996	0.83894	0.67029	0.76212
gene_len_1500_4	0.8176	0.778	0.63914	0.66907
gene_len_1500_5	0.90905	0.92857	0.77754	0.94653
gene_len_1500_6	0.91574	0.91369	0.83402	0.89104
gene_len_1500_7	0.85453	0.89123	0.62723	0.82531
gene_len_1500_8	0.8296	0.85185	0.53132	0.78894
gene_len_1500_9	0.85016	0.88977	0.65319	0.75987
gene_len_1500_10	0.78211	0.88028	0.61393	0.79779
gene_len_1500_11	0.6131	0.88028	0.5036	0.79779
gene_len_1500_12	0.91761	0.5756	0.8859	0.56603
gene_len_1500_13	0.76226	0.90734	0.64046	0.92137
gene_len_1500_14	0.70398	0.8489	0.45347	0.83778
gene_len_1500_15	0.90914	0.72235	0.838	0.66598
gene_len_1500_16	0.89139	0.8949	0.79283	0.87627
gene_len_1500_17	0.69736	0.80462	0.4893	0.81417
gene_len_1500_18	0.86071	0.80604	0.68854	0.70682
gene_len_1500_19	0.90598	0.86429	0.89946	0.83041
gene_len_1500_20	0.90662	0.91923	0.78821	0.96063
gene_len_1500_21	0.62014	0.90152	0.37076	0.8872
gene_len_1500_22	0.97455	0.77076	0.83423	0.61046
gene_len_1500_23	0.81423	0.89474	0.48896	0.9101
gene_len_1500_24	0.92286	0.92039	0.84657	0.92914
gene_len_1500_25	0.91157	0.92731	0.72334	0.92854
gene_len_1500_26	0.81726	0.88364	0.62889	0.86469
gene_len_1500_27	0.91219	0.71154	0.8781	0.65827
gene_len_1500_28	0.85464	0.92521	0.68338	0.95512
gene_len_1500_29	0.83493	0.78766	0.69357	0.76738
gene_len_1500_30	0.72607	0.9095	0.4639	0.8418

Table 6: AUC ROC and AUC for precision-recall curve for simulated data *gene_len_1500*.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
indel_no_lim_1	0.58374	0.78131	0.36792	0.70674
indel_no_lim_2	0.68041	0.71429	0.42829	0.66408
indel_no_lim_3	0.70556	0.74903	0.51088	0.74506
indel_no_lim_4	0.77355	0.81169	0.59509	0.71477
indel_no_lim_5		0.64539		0.27869
indel_no_lim_6	0.7721	0.73529	0.52128	0.60315
indel_no_lim_7	0.56807	0.77402	0.502	0.63471
indel_no_lim_8	0.71325	0.89521	0.51167	0.78327
indel_no_lim_9	0.82819	0.73077	0.71203	0.75413
indel_no_lim_10	0.75027	0.83607	0.47934	0.74162
indel_no_lim_11	0.57833	0.83607	0.39866	0.74162
indel_no_lim_12	0.92238	0.67589	0.88801	0.62463
indel_no_lim_13	0.82068	0.92532	0.79503	0.93966
indel_no_lim_14	0.74683	0.91857	0.45144	0.93348
indel_no_lim_15	0.83571	0.7998	0.68619	0.65673
indel_no_lim_16	0.89989	0.74288	0.79865	0.68751
indel_no_lim_17	0.67365	0.87225	0.4661	0.86414
indel_no_lim_18	0.90737	0.88343	0.82986	0.78286
indel_no_lim_19	0.91979	0.88636	0.9066	0.89237
indel_no_lim_20	0.91323	0.92521	0.86224	0.95512
indel_no_lim_21	0.59607	0.92468	0.45647	0.94978
indel_no_lim_22	0.89321	0.6498	0.79752	0.56021
indel_no_lim_23	0.64857	0.84921	0.38772	0.90315
indel_no_lim_24	0.92747	0.90616	0.93342	0.86901
indel_no_lim_25	0.87912	0.93155	0.61926	0.96715
indel_no_lim_26	0.84675	0.87866	0.7318	0.76519
indel_no_lim_27	0.87997	0.77137	0.61678	0.78985
indel_no_lim_28	0.89364	0.84635	0.79008	0.82463
indel_no_lim_29	0.92676	0.89643	0.91841	0.83538
indel_no_lim_30	0.71541	0.92962	0.43856	0.94908

Table 7: AUC ROC and AUC for precision-recall curve for simulated data *indel_no_lim*.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
indel_lim_5_1	0.65954	0.81621	0.44188	0.64507
indel_lim_5_2	0.64562	0.63393	0.43305	0.6225
indel_lim_5_3	0.7247	0.89423	0.63572	0.89062
indel_lim_5_4	0.70814	0.64758	0.47049	0.52971
indel_lim_5_5	0.10714	0.67992	0.34226	0.27241
indel_lim_5_6	0.83757	0.89039	0.67316	0.83706
indel_lim_5_7	0.73465	0.77069	0.52425	0.69353
indel_lim_5_8	0.84133	0.99246	0.76792	0.97368
indel_lim_5_9	0.84343	0.8033	0.72966	0.77407
indel_lim_5_10	0.81198	0.88363	0.63124	0.79842
indel_lim_5_11	0.6119	0.88363	0.56537	0.79842
indel_lim_5_12	0.90143	0.5754	0.87787	0.58988
indel_lim_5_13	0.82085	0.92143	0.65441	0.93132
indel_lim_5_14	0.78224	0.92836	0.6135	0.82781
indel_lim_5_15	0.91084	0.7906	0.76082	0.7759
indel_lim_5_16	0.90233	0.88975	0.83698	0.85153
indel_lim_5_17	0.51887	0.85408	0.28441	0.84214
indel_lim_5_18	0.89474	0.80828	0.8053	0.68657
indel_lim_5_19	0.91546	0.85053	0.90024	0.84543
indel_lim_5_20	0.89142	0.91071	0.76282	0.94383
indel_lim_5_21	0.63052	0.90901	0.4533	0.89629
indel_lim_5_22	0.86621	0.78571	0.77622	0.7095
indel_lim_5_23	0.70089	0.80102	0.60086	0.83589
indel_lim_5_24	0.92891	0.84729	0.93667	0.87399
indel_lim_5_25	0.91834	0.93027	0.74234	0.95693
indel_lim_5_26	0.87722	0.89286	0.7185	0.823
indel_lim_5_27	0.87738	0.77273	0.7136	0.76274
indel_lim_5_28	0.85935	0.91554	0.75988	0.85901
indel_lim_5_29	0.91905	0.82088	0.88226	0.78293
indel_lim_5_30	0.84319	0.90803	0.55619	0.92228

Table 8: AUC ROC and AUC for precision-recall curve for simulated data *indel_lim_5*.

Data	RAWR-AUC_ROC	Bootstrap-AUC_ROC	RAWR-AUC_PR	Bootstrap-AUC_PR
indel_pow_2_1	0.64304	0.77743	0.42251	0.61593
indel_pow_2_2	0.71672	0.72653	0.40121	0.65485
indel_pow_2_3	0.6546	0.73387	0.50989	0.60878
indel_pow_2_4	0.68885	0.61503	0.53215	0.53629
indel_pow_2_5	0.16568	0.80996	0.32326	0.60804
indel_pow_2_6	0.89139	0.9051	0.66521	0.87081
indel_pow_2_7	0.80843	0.73921	0.60696	0.55492
indel_pow_2_8	0.88376	0.90344	0.65258	0.87676
indel_pow_2_9	0.83229	0.8053	0.69091	0.76955
indel_pow_2_10	0.86969	0.78869	0.70953	0.76514
indel_pow_2_11	0.46926	0.78869	0.40798	0.76514
indel_pow_2_12	0.91281	0.62546	0.85253	0.55817
indel_pow_2_13	0.72879	0.9191	0.64358	0.91846
indel_pow_2_14	0.63222	0.85111	0.47779	0.83876
indel_pow_2_15	0.84075	0.67857	0.7045	0.64077
indel_pow_2_16	0.85476	0.8908	0.77387	0.82783
indel_pow_2_17	0.86044	0.80769	0.67249	0.80028
indel_pow_2_18	0.905	0.78499	0.78385	0.81805
indel_pow_2_19	0.92019	0.89153	0.883	0.89066
indel_pow_2_20	0.90824	0.92081	0.8271	0.95006
indel_pow_2_21	0.62268	0.91071	0.40873	0.92778
indel_pow_2_22	0.89113	0.58696	0.78344	0.57231
indel_pow_2_23	0.67437	0.84286	0.42715	0.86805
indel_pow_2_24	0.91831	0.84615	0.90184	0.738
indel_pow_2_25	0.90479	0.92115	0.65536	0.94695
indel_pow_2_26	0.85219	0.90139	0.70164	0.79055
indel_pow_2_27	0.88949	0.7432	0.66721	0.7458
indel_pow_2_28	0.84784	0.92624	0.78393	0.93505
indel_pow_2_29	0.92766	0.86646	0.92497	0.80206
indel_pow_2_30	0.76381	0.93095	0.47894	0.96301

Table 9: AUC ROC and AUC for precision-recall curve for simulated data *indel_pow_2*.