usher_wiki

Release 0.0.2

Bryan Thornlow

Jul 31, 2023

CONTENTS

Welcome to the manual for UShER package, that includes SARS-CoV-2 Phylogenetics tools UShER, matUtils, matOptimize, RIPPLES, strain_phylogenetics, and others. Please see the table of contents on the sidebar, or *click here* for a quick tutorial on getting started.

CHAPTER

ONE

QUICK START

1.1 Quick install

To quickly install the UShER package, use the following conda instructions:

```
# Create a new environment for UShER
conda create -n usher-env
# Activate the newly created environment
conda activate usher-env
# Set up channels
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
# Install the UShER package
conda install usher
```

1.2 UShER

To get acquainted with UShER, we provide a simple example of placing 10 samples on an existing phylogeny. First, download the example files.

```
wget https://raw.githubusercontent.com/yatisht/usher/master/test/global_phylo.nh
wget https://raw.githubusercontent.com/yatisht/usher/master/test/global_samples.vcf
wget https://raw.githubusercontent.com/yatisht/usher/master/test/new_samples.vcf
```

Then, create a mutation annotated tree object:

```
usher --tree global_phylo.nh --vcf global_samples.vcf --collapse-tree --save-mutation-
↪annotated-tree global_phylo.pb
```

Now, we want to place the samples from `missing_10.vcf.gz` onto our tree. We can do this by using the following command:

```
usher --vcf new_samples.vcf --load-mutation-annotated-tree global_phylo.pb --write-
↪uncondensed-final-tree
```

This yields the following three files:

- `final-tree.nh` (newick-formatted tree with identical samples condensed)
- `uncondensed-final-tree.nh` (newick-formatted tree containing all samples)

- `mutation-paths.txt` (tab-separated file containing each sample, the nodes leading to that sample in the final tree, and the mutations at those nodes)

A simplified workflow to add user-specified samples in .fasta format to the latest public MAT is available using the following commands:

```
cd usher/workflows
snakemake --use-conda --cores 4 --config FASTA="/path/to/fasta" RUNTYPE="usher"
```

1.3 matUtils

matUtils is a toolkit for rapid exploratory analysis and manipulation of mutation-annotated trees (MATs). The full manual page including detailed parameter information can be found *here*. matUtils is installed with the UShER package (see above installation instructions using conda).

```
wget https://hgwdev.gi.ucsc.edu/~angie/UShER_SARS-CoV-2/2021/05/17/public-2021-05-17.all.
↪masked.nextclade.pangolin.pb.gz
```

matUtils can quickly survey the contents of MAT files with `matUtils summary`.

```
matUtils summary -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz
```

matUtils is capable of quickly filtering on a variety of conditions and generating a series of outputs with a single call to `matUtils extract`. These outputs include newick, vcf, other pb, and Augur JSON capable of being visualized on the Auspice web platform.

```
matUtils extract -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz -a 3 -b 10 -k
↪"Scotland/QEUH-13C22D1/2021|21-03-10:100" -v my_subset.vcf -t my_subset.newick -j my_
↪subset.json
```

The above command filters samples with higher parsimony scores than 3 and ancestral branches with a greater length than 10, then collects a subtree representing the nearest 100 samples to our indicated sample. From this subtree this command generates a vcf containing all sample mutation information, a newick representing the subtree, and an Auspice-uploadable JSON in mere seconds.

Tutorials for matUtils, including an example workflow, sample placement uncertainty, and phylogeographic analysis, can be found *here*.

1.4 matOptimize

matOptimize is a program that can optimize the topology of mutation-annotated trees (MATs) using subtree pruning and regrating (SPR) moves for parsimony. The full manual page including detailed parameter information can be found *here*. matOptimize is installed with the UShER package (see above installation instructions using conda).

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/2021/05/25/
↪public-2021-05-25.all.masked.pb.gz
# matOptimize. does not support compressed files
gunzip public-2021-05-25.all.masked.pb.gz
matOptimize -i public-2021-05-25.all.masked.pb -o optimized-public-2021-05-25.all.masked.
↪pb -T 32 -r 4
```

The above commands download a public SARS-CoV-2 MAT (`public-2021-05-25.all.masked.pb.gz`) and optimize it using SPR moves of radius 4 and 32 CPU threads to produce an parsimony-optimized output MAT (`optimized-public-2021-05-25.all.masked.pb.gz`).

1.5 RIPPLES

RIPPLES is a program that can detect recombinant nodes and their ancestors in a mutation-annotated tree (MAT). The full manual page including detailed parameter information can be found *here*. RIPPLES is installed with the UShER package (see above installation instructions using conda).

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/2021/05/25/
↪public-2021-05-25.all.masked.pb.gz
ripples -i public-2021-05-25.all.masked.pb.gz -T 32 -d recomb_dir/
```

The above commands download a public SARS-CoV-2 MAT (`public-2021-05-25.all.masked.pb.gz`) and putative identify recombinant nodes using 32 CPU threads and store the results in the output directory `recomb_dir`.

CHAPTER

TWO

INSTALLATION

USHER package can be installed using three different options: (i) conda, (ii) Docker and (iii) installation scripts, as described below.

2.1 Conda

A quick method is via *conda*:

```
# Create a new environment for UShER
conda create -n usher-env
# Activate the newly created environment
conda activate usher-env
# Set up channels
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
# Install the UShER package
conda install usher
```

2.2 Conda Local Build

```
git clone https://github.com/yatisht/usher.git
cd usher/install
conda env create -f environment.yml
conda activate usher
cd ..
mkdir build
cd build
wget https://github.com/oneapi-src/oneTBB/archive/2019_U9.tar.gz
tar -xvzf 2019_U9.tar.gz
cmake  -DTBB_DIR=${PWD}/oneTBB-2019_U9  -DCMAKE_PREFIX_PATH=${PWD}/oneTBB-2019_U9/cmake .
↪.
make -j
cd ..
```

followed by, if on a MacOS system:

```
rsync -aP rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/macOSX.x86_64/faToVcf .
chmod +x faToVcf
mv faToVcf scripts/
```

or if on a Linux system:

```
rsync -aP rsync://hgdownload.soe.ucsc.edu/genome/admin/exe/linux.x86_64/faToVcf .
chmod +x faToVcf
mv faToVcf scripts/
```

Executables will be located in the build and scripts directories. Make sure they're on your path for your system as appropriate, or that you modify your commands to indicate their location.

```
export PATH=$PATH:/path/to/install/usher/build/
export PATH=$PATH:/path/to/install/usher/scripts/
```

2.3 Docker

From DockerHub:

```
docker pull pathogengenomics/usher:latest
docker run -t -i pathogengenomics/usher:latest /bin/bash
```

OR locally:

```
git clone https://github.com/yatisht/usher.git
cd usher
docker build --no-cache -t usher install/
docker run -t -i usher /bin/bash
```

2.4 Installation scripts

```
git clone https://github.com/yatisht/usher.git
cd usher
```

For MacOS 10.14 or above:

```
./install/installMacOS.sh
```

For Ubuntu 18.04 and above (requires sudo privileges):

```
./install/installUbuntu.sh
```

For CentOS 7 and above (requires sudo privileges):

```
./install/installCentOS.sh
```

CHAPTER

THREE

USHER

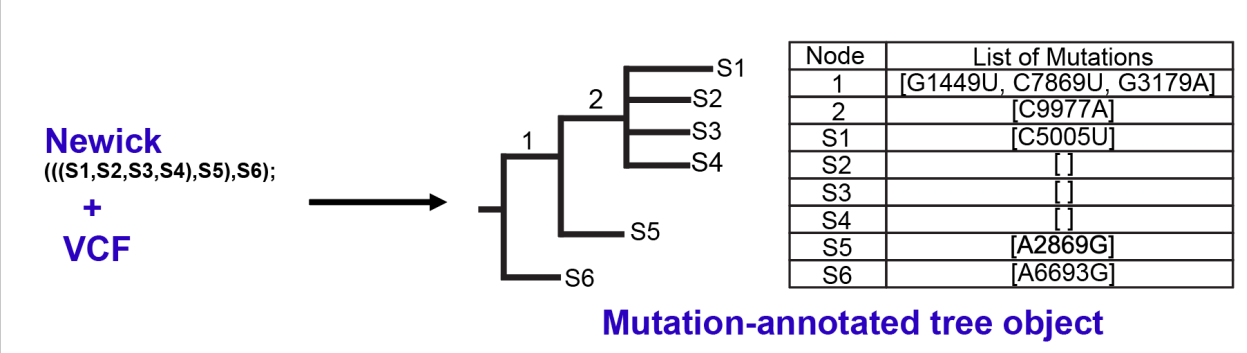
UShER is a program for rapid, accurate placement of samples to existing phylogenies. It is available for downloading here and is updated regularly. While not restricted to SARS-CoV-2 phylogenetic analyses, it has enabled real-time phylogenetic analyses and genomic contact tracing in that its placement is orders of magnitude faster and more memory-efficient than previous methods, and is being widely used by several SARS-CoV-2 research groups, including the UCSC Genome Browser team and Rob Lanfear's global phylogeny releases. We recommend using Cov2Tree develoepd by Theo Sanderson to visualize these trees.

3.1 Methodology

Given existing samples, whose genotypes and phylogenetic tree is known, and the genotypes of new samples, UShER aims to incorporate new samples into the phylogenetic tree while preserving the topology of existing samples and maximizing parsimony. UShER's algorithm consists of two phases: (i) the pre-processing phase and (ii) the placement phase.

3.1.1 Pre-processing

In the pre-processing phase, UShER accepts the phylogenetic tree of existing samples in a Newick format and their genotypes, specified as a set of single-nucleotide variants with respect to a reference sequence (UShER currently ignores indels), in a VCF format. For each site in the VCF, UShER uses the Fitch-Sankoff algorithm to find the most parsimonious nucleotide assignment for every node of the tree (UShER automatically labels internal tree nodes). When a sample contains **ambiguous genotypes**, multiple nucleotides may be most parsimonious at a node. To resolve these, UShER assigns it any one of the most parsimonious nucleotides with preference, when possible, given to the reference base. UShER also allows the VCF to specify ambiguous bases in samples using IUPAC format, which are also resolved to a unique base using the above strategy. When a node is found to carry a mutation, i.e. the base assigned to the node differs from its parent, the mutation gets added to a list of mutations corresponding to that node. Finally, UShER uses protocol buffers to store in a file, the Newick string corresponding to the input tree and a list of lists of node mutation, which we refer to as **mutation-annotated tree object**, as shown in the figure below.

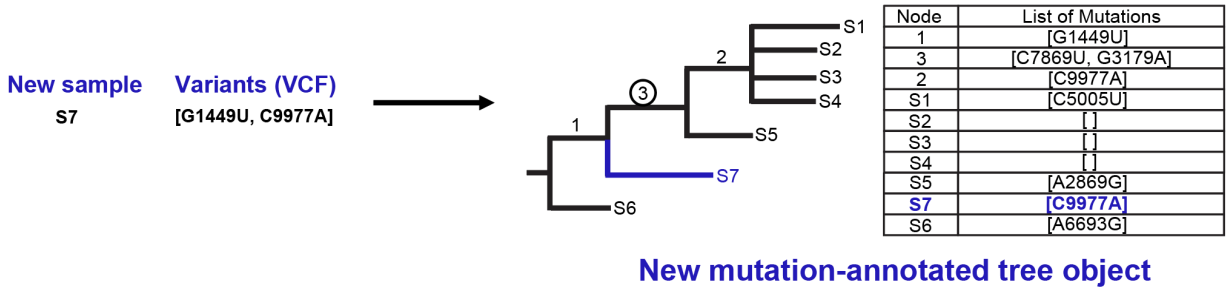


Node	List of Mutations
1	[G1449U, C7869U, G3179A]
2	[C9977A]
S1	[C5005U]
S2	[ ]
S3	[ ]
S4	[ ]
S5	[A2869G]
S6	[A6693G]

The mutation-annotated tree object carries sufficient information to derive parsimony-resolved genotypes for any tip of the tree using the sequence of mutations from the root to that tip. For example, in the above figure, S5 can be inferred to contain variants G1149U, C7869U, G3179A and A2869G with respect to the reference sequence. Compared to other tools that use full multiple-sequence alignment (MSA) to guide the placement, UShER's mutation-annotated tree object is compact and is what helps make it **fast**.

3.1.2 Placement

In the **placement phase**, UShER loads the pre-processed mutation-annotated tree object and the genotypes of new samples in a VCF format and **sequentially** adds the new samples to the tree. For each new sample, UShER computes the additional parsimony score required for placing it at every node in the current tree while considering the full path of mutations from the root of the tree to that node. Next, UShER places the new sample at the node that results in the smallest additional parsimony score. When multiple node placements are equally parsimonious, UShER picks the node with a greater number of descendant leaves for placement. If the choice is between a parent and its child node, the parent node would always be selected by this rule. However, a more accurate placement should reflect the number of leaves uniquely attributable to the child versus parent node. Therefore, in these cases, UShER picks the parent node if the number of descendant leaves of the parent that are not shared with the child node exceed the number of descendant leaves of the child. The figure below shows a new sample, S7, containing variants G1149U and C9977A being added to the previous mutation-annotated tree object in a parsimony-optimal fashion (with a parsimony score of 1 for the mutation C9977A). UShER also automatically imputes and reports **ambiguous genotypes** for the newly added samples and ignores **missing bases**, such as 'N' or '.' (i.e. missing bases never contribute to the parsimony score).



Node	List of Mutations
1	[G1449U]
3	[C7869U, G3179A]
2	[C9977A]
S1	[C5005U]
S2	[ ]
S3	[ ]
S4	[ ]
S5	[A2869G]
S7	**[C9977A]**
S6	[A6693G]

At the end of the placement phase, UShER allows the user to create another protocol-buffer (protobuf) file containing the mutation-annotated tree object for the newly generated tree including added samples as also shown in the example figure above. This allows for another round of placements to be carried out over and above the newly added samples.

3.2 Options

```
--vcf (-v): Input VCF file (in uncompressed or gzip-compressed .gz format). (REQUIRED)
--tree (-t): Input tree file.
--outdir (-d):  Output directory to dump output and log files.
--load-mutation-annotated-tree (-i): Load mutation-annotated tree object.
--save-mutation-annotated-tree (-o): Save output mutation-annotated tree to the␣
↪specified filename.
--sort-before-placement-1 (-s): Sort new samples based on computed parsimony score and␣
↪then number of optimal placements before the actual placement (EXPERIMENTAL).
--sort-before-placement-2 (-S): Sort new samples based on the number of optimal␣
↪placements and then the parsimony score before the actual placement (EXPERIMENTAL).
--reverse-sort (-r): Reverse the sorting order of sorting options (sort-before-placement-
↪1 or sort-before-placement-2). (EXPERIMENTAL)
--collapse-tree (-c): Collapse internal nodes of the input tree with no mutations and␣
↪condense identical sequences in polytomies into a single node and the save the tree to␣
↪file condensed-tree.nh in outdir.
--max-uncertainty-per-sample (-e): Maximum number of equally parsimonious placements␣
↪allowed per sample beyond which the sample is ignored. Default = 1000000.
--write-uncondensed-final-tree (-u): Write the final tree in uncondensed format and save␣
↪to file uncondensed-final-tree.nh in outdir.
--write-subtrees-size (-k): Write minimum set of subtrees covering the newly added␣
↪samples of size equal to or larger than this value. Default = 0.
--write-parsimony-scores-per-node (-p): Write the parsimony scores for adding new␣
↪samples at each existing node in the tree without modifying the tree in a file names␣
↪parsimony-scores.tsv in outdir.
--multiple-placements (-M): Create a new tree up to this limit for each possibility of␣
↪parsimony-optimal placement. Default = 1.
--retain-input-branch-lengths (-l): Retain the branch lengths from the input tree in out␣
↪newick files instead of using number of mutations for the branch lengths.
--threads (-T): Number of threads to use when possible. Default = use all available␣
↪cores.
--help (-h): Print help messages.
```

3.3 Usage

3.3.1 Display help message

To familiarize with the different command-line options of UShER, it would be useful to view its help message using the command below:

```
usher --help
```

3.3.2 Pre-processing global phylogeny

The following example command pre-processes the existing phylogeny (`global_phylo.nh`) and using the genotypes (`global_samples.vcf`) and generates the mutation-annotated tree object that gets stored in a protobuf file (`global_assignments.pb`). Note that UShER would automatically place onto the input global phylogeny any samples in the VCF (to convert a fasta sequence to VCF, consider using Fasta2USHER that are missing in the input global phylogeny using its parsimony-optimal placement algorithm. This final tree is written to a file named `final-tree.nh` in the folder specified by `--outdir` or `-d` option (if not specified, default uses current directory).

```
usher -t test/global_phylo.nh -v test/global_samples.vcf -o global_assignments.pb -d␣
↪output/
```

By default, UShER uses **all available threads** but the user can also specify the number of threads using the `--threads` or `-T` command-line parameter.

UShER also allows an option during the pre-processing phase to collapse nodes (i.e. delete the node after moving its child nodes to its parent node) that are not inferred to contain a mutation through the Fitch-Sankoff algorithm as well as to condense nodes that contain identical sequences into a single representative node. This is the **recommended usage** for UShER as it not only helps in significantly reducing the search space for the placement phase but also helps reduce ambiguities in the placement step and can be done by setting the `--collapse-tree` or `-c` parameter. The collapsed input tree is stored as `condensed-tree.nh` in the output directory.

```
usher -t test/global_phylo.nh -v test/global_samples.vcf -o global_assignments.pb -c -d␣
↪output/
```

Note the the above command would condense identical sequences, namely S2, S3 and S4, in the example figure above into a single condensed new node (named something like *node_1_condensed_3_leaves*). If you wish to display the collapsed tree without condensing the nodes, also set the `--write-uncondensed-final-tree` or `-u` option, for example, as follows:

```
usher -t test/global_phylo.nh -v test/global_samples.vcf -o global_assignments.pb -c -u -
↪d output/
```

The above commands saves the collapsed but uncondensed tree as `uncondensed-final-tree.nh` in the output directory.

3.3.3 Placing new samples

Once the pre-processing is complete and a mutation-annotated tree object is generate (e.g. `global_assignments.pb`), UShER can place new sequences whose variants are called in a VCF file (e.g. `new_samples.vcf`) to existing tree as follows:

```
usher -i global_assignments.pb -v test/new_samples.vcf -u -d output/
```

Again, by default, UShER uses **all available threads** but the user can also specify the number of threads using the *–threads* command-line parameter.

The above command not only places each new sample sequentially, but also reports the parsimony score and the number of parsimony-optimal placements found for each added sample. UShER displays warning messages if several (>=4) possibilities of parsimony-optimal placements are found for a sample. This can happen due to several factors, including (i) missing data in new samples, (ii) presence of ambiguous genotypes in new samples and (iii) structure and mutations in the global phylogeny itself, including presence of multiple back-mutations.

In addition to the global phylogeny, one often needs to contextualize the newly added sequences using subtrees of closest *N* neighbouring sequences, where *N* is small. UShER allows this functionality using `--write-subtrees-size` or `-k` option, which can be set to an arbitrary *N*, such as 20 in the example below:

```
usher -i global_assignments.pb -v test/new_samples.vcf -u -k 20 -d output/
```

The above command writes subtrees to files names `subtree-<subtree-number>.nh`. It also write a text file for each subtree (named `subtree-<subtree-number>-mutations.txt` showing mutations at each internal node of the subtree. If the subtrees contain condensed nodes, it writes the expanded leaves for those nodes to text files named `subtree-<subtree-number>-expanded.txt`.

Finally, the new mutation-annotated tree object can be stored again using `--save-mutation-annotated-tree` or `-o` option (overwriting the loaded protobuf file is allowed).

```
usher -i global_assignments.pb -v test/new_samples.vcf -u -o new_global_assignments.pb -
↪d output/
```

3.4 Features

In addition to simply placing samples on an existing phylogeny, UShER provides the user with several points of additional information, including measurements of uncertainty in sample placement, and is capable of auxiliary analyses:

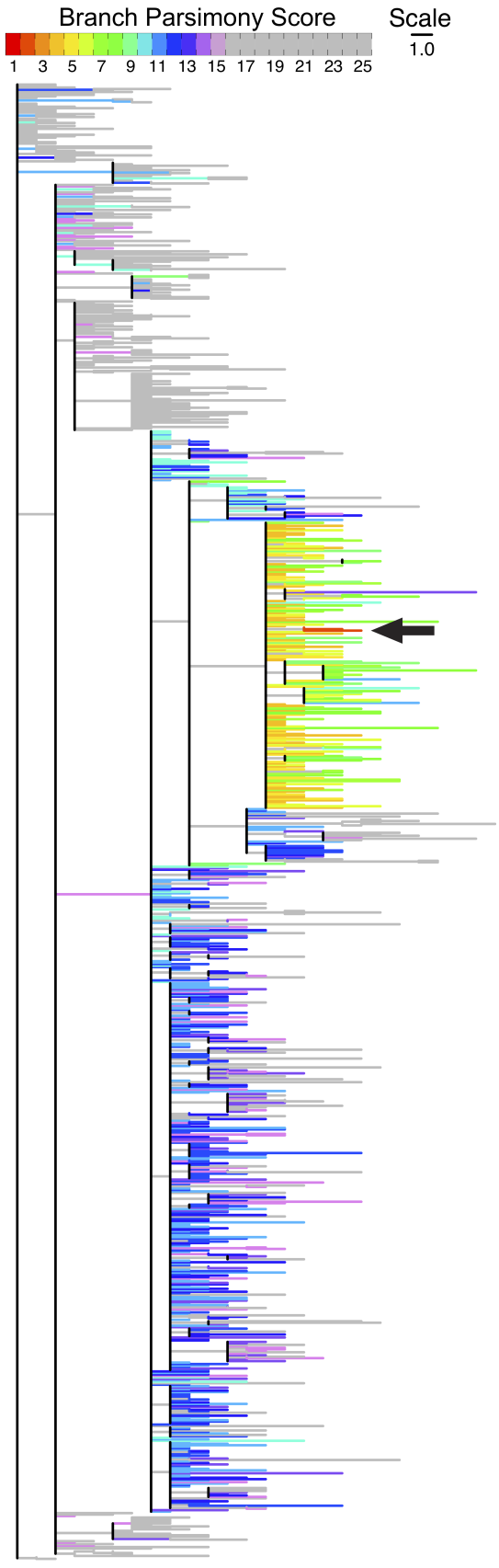
3.4.1 Branch Parsimony Score

UShER also allows quantifying the uncertainty in placing new samples by reporting the parsimony scores of adding new samples to all possible nodes in the tree **without** actually modifying the tree (this is because the tree structure, as well as number of possible optimal placements could change with each new sequential placement). In particular, this can help the user explore which nodes of the tree result in a small and optimal or near-optimal parsimony score. This can be done by setting the `--write-parsimony-scores-per-node` or `-p` option, for example, as follows:

```
usher -i global_assignments.pb -v test/new_samples.vcf -p -d output/
```

The above command writes a file `parsimony-scores.tsv` containing branch parsimony scores to the output directory. Note that because the above command does not perform the sequential placement on the tree, the number of parsimony-optimal placements reported for the second and later samples could differ from those reported with actual placements.

The figure below shows how branch parsimony score could be useful for uncertainty analysis. The figure shows color-coded parsimony score of placing a new sample at different branches of the tree with black arrow pointing to the branch where the placement is optimal. As can be seen from the color codes, the parsimony scores are low (implying good alternative placement) for several neighboring branches of the optimal branch.


Branch Parsimony Score
1 3 5 7 9 11 13 15 17 19 21 23 25
Scale
1.0

3.4.2 Multiple parsimony-optimal placements

To further aid the user to quantify phylogenetic uncertainty in placement, UShER has an ability to enumerate all possible topologies resulting from equally parsimonious sample placements. UShER does this by maintaining a list of mutation-annotated trees (starting with a single mutation-annotated tree corresponding to the input tree of existing samples) and sequentially adds new samples to each tree in the list while increasing the size of the list as needed to accommodate multiple equally parsimonious placements for a new sample. This feature is available using the `--multiple-placements` or `-M` option in which the user specifies the maximum number of topologies that UShER should maintain before it reverts back to using the default tie-breaking strategy for multiple parsimony-optimal placements in order to keep the runtime and memory usage of UShER reasonable.

```
usher -i global_assignments.pb -v <USER_PROVIDED_VCF> -M -d output/
```

Note that if the number of equally parsimonious placements for the initial samples is large, the tree space can get too large too quickly and slow down the placement for the subsequent samples. Therefore, UShER also provides an option to sort the samples first based on the number of equally parsimonious placements using the `-S` option.

```
usher -i global_assignments.pb -v <USER_PROVIDED_VCF> -M -S -d output/
```

There are many ways to interpret and visualize the forest of trees produced by multiple placements. One method is to use DensiTree, as shown using an example figure (generated using the phangorn package) below:

3.4.3 Updating multiple input trees

UShER is also fast enough to allow users to update multiple input trees incorporating uncertainty in tree resonstruction, such as multiple bootstrap trees. While we do not provide an explicit option to input multiple trees at once, UShER can be run independently for each input tree and place new samples. We recommend the user to use the GNU parallel utility to do so in parallel using multiple CPU cores while setting `-T 1` for each UShER task.

3.4.4 Finding sister clades

To determine the accuracy of each sample placement, one might be interested in knowing all of the sister clades of that sample on the final tree. We provide a utility for this calculation here. `find_sister_clades` takes the following options:

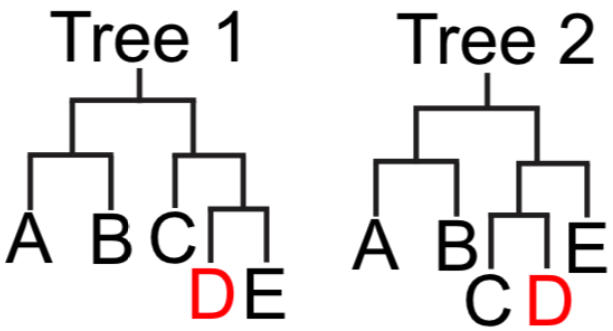
```
--tree: Input tree file.
--samples: File containing missing samples.
--generations: Number of generations.
--help: Print help messages.
```

An example usage of this function is given below:

```
find_sister_clades --generations 1 final-tree.nh --samples list-of-samples.txt > sister_
↪clades.txt
```

The samples file should have the name of each sample of interest exactly as it appears in the input tree file. The output file will contain the name of each sample with a `:`, followed by each sister sample at the input generation, separated by a new-line character. Lists of sister clades for each sample are separated by two new-lines characters.

We also described a method for measuring tree congruence involving comparing the sister clades at several generations, and finding the minimum combined number of generations at which a given sample has the same sister clades in two trees. For clarity, we provide the figure below:



N1	Tree 1 Clade	N2	Tree 2 Clade	N1+N2-2
1	DE	1	CD	N/A
		2	CDE	N/A
		3	ABCDE	N/A
2	CDE	1	CD	N/A
		2	**CDE**	**2**
		3	ABCDE	N/A
3	ABCDE	1	CD	N/A
		2	CDE	N/A
		3	**ABCDE**	**4**

3.5 Converting raw sequences into VCF for UShER input

We provide instructions below for converting raw genomic sequences in fasta format into VCF for UShER's input.

3.5.1 Generating Multiple Sequence Alignment (MSA)

Users can generate multiple sequence alignment of the input sequences using MAFFT that is already installed with UShER package. For example, we provide a number of example SARs-CoV-2 sequences in `test/Fasta2UShER` that can be combined in a single fasta file called `combined.fa` and aligned together using the SARS-CoV-2 reference sequence `test/NC_045512v2.fa` as follows:

```
cat ./test/Fasta2UShER/* > ./test/combined.fa
mafft --thread 10 --auto --keeplength --addfragments ./test/combined.fa ./test/NC_
↪045512v2.fa > ./test/myAlignedSequences.fa
```

If you have a large number of sequences, we recommend using Robert Lanfear's global_profile_alignment.sh, which can reduce memory requirements by splitting alignments and performing them in parallel.

3.5.2 Converting MSA to VCF

Users can then use the tool faToVcf, which is also installed via UShER's package, to then convert the fasta file containing multiple alignment of input sequences into a VCF. If the file `./test/myAlignedSequences.fa` includes the reference sequence (for SARS-CoV-2, NC_045512.2) as its first sequence, then faToVcf can be run like this:

```
faToVcf ./test/myAlignedSequences.fa ./test/test_merged.vcf
```

If the reference sequence is included in `./test/myAlignedSequences.fa` but is not the first sequence, then its name needs to be specified using the `-ref=...` option:

```
faToVcf -ref=NC_045512.2 ./test/myAlignedSequences.fa ./test/test_merged.vcf
```

If the reference sequence is not included in `./test/myAlignedSequences.fa` then it can be added in the bash shell using a named pipe like this:

```
faToVcf <(cat NC_045512.2.fa ./test/myAlignedSequences.fa) ./test/test_merged.vcf
```

For SARS-CoV-2 data, we recommend downloading `problematic_sites_sarsCov2.vcf` and using it for masking problematic sites as follows:

```
wget https://raw.githubusercontent.com/W-L/ProblematicSites_SARS-CoV2/master/problematic_
↪sites_sarsCov2.vcf
faToVcf -maskSites=problematic_sites_sarsCov2.vcf ./test/myAlignedSequences.fa ./test/
↪test_merged_masked.vcf
```

The resulting VCF files `test_merged.vcf` and `test_merged_masked.vcf` from the above commands should be compatible with UShER.

3.6 Presentations

Russ Corbett-Detig has created a module on UShER for the CDC:

Yatish Turakhia has presented on UShER at the Covid-19 Dynamics & Evolution Meeting, held virtually on October 19-20, 2020. You can find his slides here.

3.7 Publications

- Turakhia Y, Thornlow B, Hinrichs A, De Maio N, Gozashti L, Lanfear R, Haussler D, and Corbett-Detig R. Ultrafast Sample placement on Existing tRees (UShER) enables real-time phylogenetics for the SARS-CoV-2 pandemic., *Nature Genetics*. 2021. 1-8.

CHAPTER

FOUR

MATUTILS

matUtils is a suite of tools used to analyze, edit, and manipulate mutation annotated tree (.pb) files, such as the ones shared in our public SARS-CoV-2 MAT database (http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/) or those constructed via UShER (read *this* and *this* for steps to construct a MAT from an input phylogeny and an alignment).

4.1 Installation

To install matUtils, simply follow the directions for *installing UShER*, and matUtils will be included in your installation.

4.2 The Mutation Annotated Tree (MAT) Protocol Buffer (.pb)

Google's protocol buffer format is a highly optimized, flexible binary storage format, with APIs for many languages. We use a specially formatted protocol buffer to store a Mutation Annotated Tree object. The .proto definition file can be found in the Usher installation folder; we will describe it in brief here.

Our protobuf format has two major components. The first is a newick string, representing phylogenetic relationships between all samples included in the tree. The second component is node information- including mutations, clade assignments, and condensed nodes. These components combined make the MAT into a powerful data storage format, as it efficiently simultaneously represents phylogenetic relationships and full mutation information for large numbers of samples.

When the protobuf is loaded into matUtils, the tree is structured based on the stored newick string, then mutations and metadata are placed along the tree structure according to the node information vectors. The condensed nodes message is used to compact polytomies and other groups of identical samples, and the final step of loading is to uncondense these nodes back into full polytomies.

This structure allows us to perform both classical phylogenetic processes, such as traversing the tree and calculating parsimony scores, while also being able to query the structure as if it was a database of sequences, extracting a vcf of samples which contain a specific mutation for example.

4.3 matUtils Common Options

All matUtils subcommands include these parameters.

```
--input-mat (-i): Input mutation-annotated tree file. (REQUIRED)
--threads (-T): Number of threads to use when possible. Default = use all available␣
↪cores.
--help (-h): Print help messages.
```

4.4 summary

matUtils summary is used to get basic statistics and attribute information about the mat. If no specific arguments are set, prints the number of nodes, number of samples, number of condensed nodes, and total tree parsimony of the input mat to standard output.

Amino acid translations

`matUtils summary --translate <output.tsv> -i <input.pb> -g <annotations.gtf> -f <reference.fasta>`

performs phylogenetically informed annotation of amino acid mutations.

The user provides as input a protobuf file, a GTF file containing gene annotations, and a FASTA reference sequence.

Note: The input GTF must follow the conventions specified here. If multiple `CDS` features are associated with a single `gene_id`, they must be ordered by start position. An example GTF for SARS-CoV-2 can be found here.

The output format is a TSV file with four columns, with one line per node (only including nodes with mutations), e.g.

node_id	aa_mutations	nt_mutations	codon_changes	leaves_sharing_mutations
Sample1	ORF7a:E121D;ORF7b:M1L	A27756T;A27756T	GAG>GTG;GAG>GTG	1
Sample2	S:R905R	G24277A	GGG>AGG	1
Sample5	S:Y756N	T23828A	CCT>CCA	1
node_2	M:V60G;M:V66A	T26701G;T26719C	GTA>GGA;ACT>ACC	2
Sample4	M:G60R;M:A66V	G26700C;C26719T	GTA>CTA;ACC>ACT	1
Sample3	M:T30A	A26610G	TTA>TTG	1

`aa_mutations` are always delimited by a `;` character, and can be matched with their corresponding nucleotide mutations in the `nt_mutations` column (also delimited by `;`). Codon changes are encoded similarly.

If there are multiple nucleotide mutations in one node affecting a single codon (rare), they will be separated by commas in the `nt_mutations` column.

In the case that a single nucleotide mutation affects multiple codons, the affected codons are listed sequentially, and the nucleotide mutation is repeated in the `nt_mutation` column.

`leaves_sharing_mutations` indicates the number of descendant leaves of the node that share its set of mutations (including itself, if the node is a leaf).

RoHo score

Additionally, *matUtils summary* can quickly calculate the RoHo score and related values described in van Dorp et al 2020. Briefly, the RoHo or Ratio of Homoplasic Offspring is the ratio of the number of descendents in sister clades with or without a specific mutation over the occurrence of all mutations; homoplasic and positively-selected mutations

will recur with increased descendent clade sizes at each occurrence. This can be used to quickly and conservatively scan for variants of concern. A full explanation of our implementation and tutorial can be found *here.*

4.4.1 Example Usage

1. Get a tsv containing all sample names and parsimony scores.

```
matUtils summary -i input.pb --samples samples.txt
```

```
matUtils summary -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz --samples 06-
↪09_samples.txt
```

2. Write all possible summary output files to a specific directory.

```
matUtils summary -i input.pb -A -d input_summary/
```

3. Get amino acid translations of each node in a tree

```
matUtils summary -t translate_output.tsv -i input.pb -g annotation.gtf -f reference.fasta
```

Example command

Files needed:

a. public-2021-06-09.all.masked.nextclade.pangolin.pb.gz

```
matUtils summary -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -A -d 06-09_
↪summary/
```

4.4.2 Specific Options

```
--input-mat (-i): Input mutation-annotated tree file [REQUIRED]. If only this argument␣
↪is set, print the count of samples and nodes in the tree.
--input-gtf (-g): Input GTF annotation file. Required for --translate (-t)
--input-fasta (-g): Input FASTA reference sequence. Required for --translate (-t)
--output-directory (-d): Write all output files to the target directory. Default is␣
↪current directory
--samples (-s): Write a two-column tsv listing all samples in the tree and their␣
↪parsimony score (terminal branch length). Auspice-compatible.
--clades (-c): Write a tsv listing all clades and the count of associated samples in the␣
↪tree.
--mutations (-m): Write a tsv listing all mutations in the tree and their occurrence␣
↪count.
--translate (-t): Write a tsv listing the amino acid and nucleotide mutations at each␣
↪node.
--aberrant (-a): Write a tsv listing potentially problematic nodes, including duplicates␣
↪and internal nodes with no mutations and/or branch length 0.
--haplotype (-H): Write a tsv listing haplotypes represented by comma-delimited lists of␣
↪mutations and their count across the tree.
--sample-clades (-C): Write a tsv listing all samples and their clades.
--calculate-roho (-R): Write a tsv listing, for each mutation occurrence that is valid,␣
↪the number of offspring and other numbers for RoHo calculation.
```

(continues on next page)

(continued from previous page)

```
--expanded-roho (-E): Use to include date and other contextual information in the RoHO␣
↪output. Significantly slows calculation time.
--get-all (-A): Write all possible tsv outputs with default file names (samples.txt,␣
↪clades.txt, etc).
```

4.5 extract

matUtils extract serves as a flexible prebuilt pipeline, and serves as the primary tool for subsetting and converting a MAT pb to other file formats. Generally, its parameters can be grouped into four categories:

1. Selection- these parameters define a set of samples constituting a subtree to use for downstream analysis. If none are set, the whole input tree is used. Includes -c, -s, -m, and others.
2. Processing- these parameters, usually boolean, apply specific processing steps to the subtree after sample selection. These can include selecting clade representative samples or collapsing the tree. Includes -O and -r among others.
3. Information- these parameters save information about the subtree which is not a direct representation of that subtree, such as mutations defining each clade in the subtree. Includes -C and -S among others.
4. Conversion- these parameters are used to request subtree representations in the indicated formats. Includes -v and -t among others.

matUtils extract is the workhorse function for manipulating MAT .pb files, particularly for any operations involving removing part of the .pb and converting .pb to other file formats.

4.5.1 Example Syntax and Usage

1. Write a vcf representing all samples within a clade.

```
matUtils extract -i input.pb -c my_clade -v my_clade.vcf
```

```
matUtils extract -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -c B.1.351 -v␣
↪351_samples.vcf
```

The VCF file can be converted to a Fasta files (one for each sequence in the VCF) using vcf2fasta (see installation instructions here) and the reference sequence (in NC_045512v2.fa) as follows:

```
vcfindex 351_samples.vcf
vcf2fasta -f NC_045512v2.fa 351_samples.vcf
```

Note that indels are ignored in the above approach since they're not included in the MAT.

2. Write a newick tree of all samples which contain either of two mutations of interest.

```
matUtils extract -i input.pb -m my_mutation,my_other_mutation -t my_mutations.nwk
```

```
matUtils extract -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -m G7328T,
↪A8653G -t double_mutant.nwk
```

3. Convert a MAT JSON into a .pb file, while removing branches with length greater than 7.

```
matUtils extract -i input.json -b 7 -o filtered.pb
```

```
matUtils extract -i usa_group.json -b 7 -o filtered_usa_group.pb
```

4. Generate a MAT JSON representing a subtree of size 25 around a sample of interest, including multiple metadata files and filtering low-scoring samples.

```
matUtils extract -i input.pb -a 5 -M my_metadata_1.tsv,my_metadata_2.tsv -k my_sample:25␣
↪-j my_sample_context.json
```

Example command

Files needed:

a. public-2021-06-09.all.masked.nextclade.pangolin.pb.gz

b. `usa_group.json`

c. `region.tsv`

d. `misc.tsv`

```
matUtils extract -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -a 5 -M meta_1.
↪tsv,meta_2.tsv -k "Scotland/CVR6436/2020|2020-12-30:25" -j cluster.json
```

4.5.2 Specific Options

```
--input-mat (-i): For this specific command, the input can either be a standard MAT␣
↪protobuf or an Augur-v2-formatted MAT JSON, ala Nextstrain.
--input-gtf (-g): Input GTF annotation file. Required for --write-taxodium.
--input-fasta (-f): Input FASTA reference sequence. Required for --write-taxodium.
--samples (-s): Select samples by explicitly naming them, one per line in a plain text␣
↪file.
--metadata (-M): Comma delimited names of tsvs or csvs containing sample identifiers in␣
↪the first column and an arbitrary number of metadata values in separate columns,␣
↪including a header line in each file. Used only with -j and -K.
--clade (-c): Select samples by membership in any of the indicated clade(s), comma␣
↪delimited- e.g. -c clade1,clade2.
--mutation (-m): Select samples by whether they contain any of the indicated mutation(s),
↪ comma delimited- e.g. -m mutation1,mutation2.
--match (-H): Select samples by whether their identifier matches the indicated regex␣
↪pattern.
--max-epps (-e): Select samples by whether they have less than or equal to the maximum␣
↪number of indicated equally parsimonious placements. Explanation of equally␣
↪parsimonious placements is here INSERT LINK.
--max-parsimony (-a): Select samples by whether they have less than or equal to the␣
↪indicated maximum parsimony score (terminal branch length).
--max-branch-length (-b): Remove samples which have branches of greater than the␣
↪indicated length in their ancestry.
--max-path-length (-P): Select samples which have a total path length (number of␣
↪mutations different from reference) less than or equal to the indicated value.
--nearest-k (-k): Select a specific sample and X context samples, formatted as "sample_
↪name:X".
--nearest-k-batch (-K): Pass a text file of sample IDs and a number of the number of␣
```

(continues on next page)

(continued from previous page)

```
↪context samples, formatted as sample_file.txt:k. These will be automatically written␣
↪to a series of json files named "*sample-name*_context.json". Used for special large-
↪scale operations.
--get-internal-descendents (-I): Select the set of samples descended from the indicated␣
↪internal node.
--from-mrca (-U): Select all samples which are descended from the most recent common␣
↪ancestor of the indicated set of samples. Applied before filling background with␣
↪random samples.
--set-size (-z): Automatically add or remove samples at random from the selected sample␣
↪set until it is the indicated size.
--limit-to-lca (-Z): Use to limit random samples chosen with -z or -W to below the most␣
↪recent common ancestor of all other samples.
--get-representative (-r): Toggle to automatically select two representative samples per␣
↪clade currently included in the tree, pruning all other samples from the tree. Applies␣
↪after other selection steps.
--prune (-p): Toggle to instead exclude all indicated samples from the subtree output.
--resolve-polytomies (-R): Toggle to resolve all polytomies by assigning new internal␣
↪nodes with branch length 0. Used for compatibility with other software.
--output-directory (-d): Write all output files to the target directory. Default is␣
↪current directory.
--used-samples (-u): Write a simple text file containing selected sample names.
--sample-paths (-S): Write the path of mutations defining all samples in the subtree to␣
↪the indicated file.
--clade-paths (-C): Write the path of mutations defining each clade in the subtree to␣
↪the indicated file.
--all-paths (-A): Write the mutations assigned to each node in the subtree in depth-
↪first traversal order to the indicated file.
--write-vcf (-v): Write a VCF representing the selected subtree to the target file.
--no-genotypes (-n): Do not include sample genotype columns in VCF output. Used only␣
↪with --write-vcf.
--write-mat (-o): Write the selected subtree as a new protobuf file to the target file.
--collapse-tree (-O): Collapse the MAT before writing it to protobuf output. Used only␣
↪with the write-mat option.
--write-json (-j): Write an Auspice-compatbile json representing the selected subtree.
--retain-branch-length (-E): Use to not recalculate branch lengths with saving newick␣
↪output. Used only with -t
--write-tree (-t): Write a newick string representing the selected subtree to the target␣
↪file.
--write-taxodium (-l): Write a taxodium-format protobuf to the target file.
--x-scale (-G): Specifies custom X-axis scaling for Taxodium output. Does not affect␣
↪other output formats.
--title (-B): Title to include in --write-taxodium output.
--description (-D): Description to include in --write-taxodium output.
--include-nt (-J): Include nucleotide changes in the taxodium output.
--extra-fields (-F): Comma delimited list of additional fields to include in --write-
↪taxodium output.
--minimum-subtrees-size (-N): Use to generate a series of JSON or Newick format files␣
↪representing subtrees of the indicated size covering all queried samples. Uses and␣
↪overrides -j and -t output arguments.
--reroot (-y): Indicate an internal node ID to reroot the output tree to. Applied before␣
↪all other manipulation steps.
--usher-single-subtree-size (-X): Use to produce an usher-style single sample subtree of␣
```

(continues on next page)

(continued from previous page)

```
↪the indicated size with all selected samples plus random samples to fill. Produces .nh␣
↪and .txt files in the output directory.
--usher-minimum-subtrees-size(-x): Use to produce an usher-style minimum set of subtrees␣
↪of the indicated size which include all of the selected samples. Produces .nh and .txt␣
↪files in the output directory.
--usher-clades-txt: Use to write an usher-style clades.txt alongside an usher-style␣
↪subtree with -x or -X.
--add-random (-W): Add exactly W samples to your selection at random. Affected by -Z and␣
↪overridden by -z.
--closest-relatives (-V): Write a tsv file of the closest relative(s) (in mutations) of␣
↪each selected sample to the indicated file. All equidistant closest samples are␣
↪included unless --break-ties is set.
--break-ties (-q): Only output one closest relative per sample (used only with --closest-
↪relatives). If multiple closest relatives are equidistant, the lexicographically␣
↪smallest sample ID is chosen.
--select-nearest (-Y): Set to add to the sample selection the nearest Y samples to each␣
↪of your samples, without duplication.
--dump-metadata (-Q): Set to write all final stored metadata as a tsv.
--whitelist (-L): Pass a list of samples, one per line, to always retain in the output␣
↪regardless of any other parameters.
```

4.6 annotate

matUtils annotate is a function for adding clade annotation information to the pb. This information can be accessed downstream by *matUtils extract* or other tools.

It has two general ways to add this information. The first, recommended fashion is to pass node identifiers to be be assigned as clade roots for each clade directly through a two-column tsv.

With the second method, the user can provide a set of names for the representative sequences for each clade in a two-column tsv from which the clade roots can be automatically inferred. Specifically, *matUtils annotate* expects clades in the first column and sample identifiers in the second, as in the lineageToPublicName files available in our database. The algorithm is described below:

1. It collects the set of mutations which are at at least -f allele frequency across the input samples; these represent the clade's likely defining mutations.
2. It creates a virtual sample from these mutations and uses Usher's highly optimized mapping algorithm to identify the internal nodes that it maps to best. These are candidate clade root nodes. Nodes directly ancestral to these sites are also considered as candidate roots.
3. For each of the candidate roots, the algorithm calculates the number of samples which are actually descendent of that node. The node which is ancestral to the most samples and which is not already assigned to another clade is assigned as the best clade root.

This method does not guarantee that every sample that is member in the clade in the input will be a member of the clade at the end of assignment, but assignments are generally high quality.

4.6.1 Example Syntax and Usage

1. Assign a new set of custom clade annotations to the tree.

```
matUtils annotate -i input.pb -c my_clade_info.txt -o annotated.pb
```

Example command

Files needed:

a. public-2021-06-09.all.masked.pb.gz

b. cladeToPublicName.gz

```
gunzip -c cladeToPublicName.gz > clade_info.txt
matUtils annotate -i public-2021-06-09.all.masked.pb.gz -c clade_info.txt -o nxts_
↪annotated.pb
```

4.6.2 Specific Options

```
--output-mat (-o): Path to output processed mutation-annotated tree file (REQUIRED)
--clade-names (-c): Path to a file containing clade asssignments of samples. An␣
↪algorithm automatically locates and annotates clade root nodes.
--clade-to-nid (-C): Path to a tsv file mapping clades to their respective internal node␣
↪identifiers. Use with caution.
--clade-paths (-P): Path to a tsv file mapping clades to mutation paths which must exist␣
↪in the tree.  Format is the same as the first and third columns of the output of␣
↪matUtils extract --clade-paths.
--clade-mutations (-M): Path to a tsv file mapping clades to sets of mutations␣
↪(separated by spaces,commas and/or >s) which will be used instead of extracting␣
↪mutations from samples named in the --clade-names file.  If used together with --clade-
↪names, this takes precedence.
--allele-frequency (-f): Minimum allele frequency in input samples for finding the best␣
↪clade root. Used only with -l. Default = 0.8.
--mask-frequency (-m): Minimum allele frequency below -f in input samples that should be␣
↪masked for finding the best clade root. Used only with -c.
--clip-sample-frequency (-p): Maximum proportion of samples in a branch that are␣
↪exemplars from -c to consider when sorting candidate clade root nodes. Default 0.1
--set-overlap (-s): Minimum fraction of the lineage samples that should be desecendants␣
↪of the assigned clade root. Defualt = 0.6.
--clear-current (-l): Use to remove current annotations before applying new annotations.
--output-directory (-d): Write output files to the target directory.
--write-mutations (-u): Write a tsv listing each clade and the mutations found in at␣
↪least [-f] of the samples. Used only with -c.
--write-details (-D): Write a tsv with details about the nodes considered for each clade␣
↪root. Used only with -c
```

4.7 uncertainty

matUtils uncertainty calculates two specific metrics for sample placement certainty. These metrics can be very important to support contact tracing and reliable identification of the origin of a newly placed sample.

The first of these is "equally parsimonious placements" (EPPs), which is the number of places on the tree a sample could be placed equally well. An EPPs score of 1 is a "perfect score", indicating that there is a single best placement for this sample on the tree. About 85% of samples on a normal SARS-COV-2 phylogeny have an EPPs of 1.

matUtils uncertainty calculates this metric by, for each sample in the input, remapping the sample against the same tree (disallowing it from mapping to itself) with Usher's optimized mapper function. This function reports the number of best placements as part of the output, which is recorded by *matUtils uncertainty* and saved to the text file.

The second metric is "neighborhood size score" (NSS), which is the longest direct traversable path between any two equally parsimonious placement locations for a given sample. This metric is complementary to EPPs. When EPPs is 1, NSS is necessarily 0, as there are no traversable paths between pairs of placements when there's only one placement.

On an intuitive level, NSS is a representation of the distribution of equally parsimonious placements. For example, lets say we have two samples of interest. The first has five equally parsimonious placements, but they're all quite nearby each other on the tree with the LCA two nodes back. The second has two equally parsimonious placements, but they're on opposite sides of the tree with the LCA at the root. If we only looked at EPPs, we might assume that the second sample is more certain than the first. This is absolutely not the case- the second sample could have originated from two different continents, while the first is likely from a specific local region. This is reflected in their NSS, which in the first case should be less than five, but in the latter case could be in the dozens.

The most confident samples are ones which have an EPPs of 1 and an NSS of 0, followed by ones with low EPPS values and low NSS, followed by ones with higher EPPS and low NSS, and finally ones that are high on both metrics.

NSS is calculated by taking the set of equally parsimonious placements indicated by Usher's mapper function and identifying the LCA of all placements. The two longest distances from the LCA to two placements are then summed and the result is reported as NSS- the longest direct path between two placements for the sample.

An example workflow for calculating and visualizing uncertainty metrics can be found *here*.

4.7.1 Example Syntax and Usage

1. Calculate uncertainty metrics for a specific set of samples.

```
matUtils uncertainty -i input.pb -s my_samples.txt -e my_uncertainty.tsv
```

Example command

Files needed:

a. public-2021-06-09.all.masked.nextclade.pangolin.pb.gz

b. `eng_samples.txt`

```
matUtils uncertainty -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -s eng_
↪samples.txt -e eng_uncertainty.tsv
```

4.7.2 Options

```
--samples (-s): File containing samples to calculate metrics for.
--find-epps (-e): Writes an Auspice-compatible two-column tsv of the number of equally␣
↪parsimonious placements and neighborhood sizes for each sample to the target file.
--record-placements (-o): Name for an Auspice-compatible two-column tsv which records␣
↪potential parents for each sample in the query set.
--dropout-mutations (-d): Name a file to calculate and store locally-recurrent mutations␣
↪which may be associated with primer dropout. [EXPERIMENTAL]
```

4.8 introduce

matUtils introduce is used aid the analysis of the number of new introductions of the virus genome into a geographic region. *matUtils introduce* can calculate the association index (Wang et al. 2001) or the maximum monophyletic clade size statistic (Salemi et al. 2005; Parker et al. 2008) for arbitrary sets of samples, and also uses a new heuristic (currently experimental and described below) for estimating the points of introduction.

It requires a two-column tsv as input alongside the protobuf containing names of samples and associated regions in the first and second columns respectively. Multiple regions can be processed simultaneously; in this mode, introduction points will be checked for whether they have significant support for originating from another input region.

matUtils introduce uses date ranges to sort the output so that the largest clusters occurring in the shortest timespan are printed first. Date can be automatically parsed from the sample name if the sample has standard formatting or passed in as a separate file with -M. In the latter case, the first row should be a header containing "sample_id" and "date" (other columns will be ignored). Each date included should be formatted as XXXX-XX-XX with zeroes filled in as needed (e.g. 2021-01-03) and samples excluded from this file but included in the sample region input (-s) will have their date ignored for the purposes of establishing active date ranges for their cluster.

An example workflow for inferring and visualizing geographic introductions can be found *here*.

4.8.1 Heuristic

The heuristic we use is a confidence metric which weights both the number and distance to the nearest descendent leaf which is a member of the input region to infer whether each internal node is likely to represent sequences from that region. When the confidence metric is greater than 0.5, it is considered to be in the region.

Leaves have their confidence heuristic as either 0 or 1 based on whether they are included in the regional input list. Internal nodes are assigned as in-region if all descendent leaves are in-region, and similarly out if all descendent leaves are out-region. If their descendents are a mix of samples, we use the following calculation:

Do = the distance in mutations to the nearest descendent leaf that is not in region

Di = the distance in mutations to the nearest descendent leaf that is in the region

No = the number of leaf descendents which are not in the region

Ni = the number of leaf descendents which are in the region

confidence = 1 / (1 + ((Di/Ni)/(Do/No)))

This is essentially a ratio placed under a squash function such that equal numbers of leaves and distance to the nearest leaf for both in and out of the region yield a confidence of 0.5, while descendents nearly being purely either in or out of the region will yield ~1 and ~0 respectively.

Introduction points are identified as the point along a sample's history where the confidence of the relevant node being in the region drops below 0.5. In many sample's cases, this may be the direct parent of the sample, implying that the

sample is a novel introduction to a region; in other cases, it may share the introduction point with a number of other samples from that region.

Introductions are calculated for each region independently with multiple region input, but after introductions are identified the support for the origin point for membership in each other region is checked. Origins with a confidence of >0.5 membership in other regions are recorded in the output, and if none are found the origin is labeled as indeterminate.

matUtils introduce yields a by-sample output table which includes cluster level statistics in the first few columns. Notable among these is cluster growth score, which is simply the number of samples from the cluster divided by one plus the number of weeks separating the oldest and most recent samples of the cluster, rounded down. Clusters are printed in order of their rank by this score, putting putative outbreaks and groups of most concern at the top of the output file.

4.8.2 Phylogeographic Statistics and Other Options

matUtils introduce supports the calculation and recording of maximum monophyletic clade size and association index, statistics for phylogeographic trait association, on a per-region and per-introduction basis. Maximum monophyletic clade size is simply the largest monophyletic clade of samples which are in the region; it is larger for regions which have relatively fewer introductions per sample and correlates with overall sample number (Parker et al 2008). Association index is a more complex metric, related to our heuristic, which performs a weighted summation across the tree account for the number of child nodes and the frequency of the most common trait (Wang et al 2001). Association index is smaller for stronger phylogeographic association; it increases with the relative number of introductions into a region. For association index, *matUtils introduce* also performs a series of permutations to establish an expected range of values for the random distribution of samples across the tree.

Most proper regions will have association indeces significantly smaller than this range; the ratio between the actual and mean expected association indices can be informative for the overall level of isolation or relative level of community spread for a region.

Calculating these statistics adds significantly to runtime, so they are optional to calculate and intended for users who want a stronger statistical grounding for their results.

Finally, it supports inference of the region of origin for all annotated clade roots currently in the tree based on these confidence metrics, though only from among input regions.

Wang, T.H., Donaldson, Y.K., Brettle, R.P., Bell, J.E., and Simmonds, P. (2001). Identification of Shared Populations of Human Immunodeficiency Virus Type 1 Infecting Microglia and Tissue Macrophages outside the Central Nervous System. Journal of Virology 75, 11686–11699.

Salemi M, Lamers SL, Yu S, de Oliveira T, Fitch WM, McGrath MS. 2005. Phylodynamic Analysis of Human Immunodeficiency Virus Type 1 in Distinct Brain Compartments Provides a Model for the Neuropathogenesis of AIDS. J Virol 79:11343–11352.

Parker, J., Rambaut, A., and Pybus, O.G. (2008). Correlating viral phenotypes with phylogeny: Accounting for phylogenetic uncertainty. Infection, Genetics and Evolution 8, 239–246.

4.8.3 Example Syntax and Usage

1. Generate a tsv containing inferred introduction information, one sample per row.

```
matUtils introduce -i public.pb -s my_region_samples.txt -o my_region_introductions.tsv
```

Example command

Files needed:

a. public-2021-06-09.all.masked.nextclade.pangolin.pb.gz

b. regional-samples.txt

```
matUtils introduce -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -s regional-
↪samples.txt -o regional-introductions.tsv
```

4.8.4 Options

```
--population-samples (-s): Two-column tab-separated text file containing sample names␣
↪and region membership respectively (REQUIRED)
--output (-o): Name of the output tab-separated table containing inferred introductions,␣
↪one sample per row (REQUIRED)
--additional-info (-a): Use to calculate additional phylogeographic statistics about␣
↪your region and inferred introductions.
--clade-regions (-c): Set to optionally write a tab-separated table containing inferred␣
↪origins for each clade currently annotated in the tree from among your regions.
--date-metadata (-M): Pass a TSV or CSV containing a 'date' column to use for date␣
↪information. If not used, date will be inferred from the sample name where possible.
--origin-confidence (-C): Set to a confidence value between 0 and 1 at which to state␣
↪that a node is in-region. Default is 0.5
--evaluate-metadata (-E): Set to assign each leaf a confidence value based on distance-
↪weighted ancestor confidence.
--dump-assignments (-D): Indicate a directory to which to write two-column text files␣
↪containing node-confidence assignments for downstream processing.
--cluster-output (-u): Write a one-cluster-per-row version of the output table to the␣
↪indicated file.
--latest-date (-l): Limit reported clusters to ones with at least one sample past the␣
↪indicated date.
--earliest-date (-L): Limit reported clusters to ones with ALL samples past the␣
↪indicated date.
--num-to-report (-r): Report the top r scoring potential origins for each cluster. Set␣
↪to 0 to report all passing baseline.
--minimum-to-report (-R): Report only potential origins with at least the indicated␣
↪confidence score.
--minimum-gap (-G): The minimum number of mutations between the last ancestor inferred␣
↪to be in region to its parent to use the ancestor to define the cluster instead of the␣
↪parent. Set to higher values to merge sibling clusters. Default 0.
--num-to-look (-X): Additionally require that the next X nodes on the path to the root␣
↪from the putative introduction point have lower confidences than the introduction␣
↪point. Set to higher numbers to merge nested clusters. Default 0.
```

4.9 Publications

- McBroome J, Thornlow B, Hinrichs AS, Kramer A, De Maio N, Goldman N, Haussler D, Corbett-Detig R, and Turakhia Y. A daily-updated database and tools for comprehensive SARS-CoV-2 mutation-annotated trees., *Molecular Biology and Evolution*. 2021.

CHAPTER

FIVE

MATOPTIMIZE

matOptimize is a program used to optimize phylogenies using parsimony score. It is used on MAT files, which can be created by UShER.

5.1 Installation

To install matOptimize, simply follow the directions for *installing UShER*, and matOptimize will be included in your installation.

5.2 Options

```
-v [ --vcf ] arg                      Input VCF file (in uncompressed or
                                       gzip-compressed .gz format)
 -t [ --tree ] arg                     Input tree file
 -T [ --threads ] arg (=12)            Number of threads to use when possible
                                       [DEFAULT uses all available cores, 12
                                       detected on this machine]
 -i [ --load-mutation-annotated-tree ] arg
                                       Load mutation-annotated tree object
 -o [ --save-mutation-annotated-tree ] arg
                                       Save output mutation-annotated tree
                                       object to the specified filename
                                       [REQUIRED]
 -r [ --radius ] arg (=-1)             Radius in which to restrict the SPR
                                       moves.
 -S [ --profitable-src-log ] arg (=/dev/null)
                                       The file to log from which node a
                                       profitable move can be found.
 -a [ --ambi-protobuf ] arg            Continue from intermediate protobuf
 -s [ --minutes-between-save ] arg (=0)
                                       Minutes between saving intermediate
                                       protobuf
 -m [ --min-improvement ] arg (=0.000500000024)
                                       Minimum improvement in the parsimony
                                       score as a fraction of the previous
                                       score in ordder to perform another
                                       iteration.
 -d [ --drift_iteration ] arg (=0)     Iterations permiting equally
```

(continues on next page)

(continued from previous page)

```
                                    parsimonious moves after parsimony
                                    score no longer improves
-n [ --do-not-write-intermediate-files ]
                                    Do not write intermediate files.
-N [ --max-iterations ] arg (=1000)  Maximum number of optimization
                                    iterations to perform.
-M [ --max-hours ] arg (=0)         Maximium number of hours to run
-V [ --transposed-vcf-path ] arg    Auxiliary transposed VCF for ambiguous
                                    bases, used in combination with usher
                                    protobuf (-i)
--version                           Print version number
-z [ --node_proportion ] arg (=2)   the proportion of nodes to search
-y [ --node_sel ] arg               Random seed for selecting nodes to
                                    search
-h [ --help ]                       Print help messages
```

5.3 Presentations

Cheng Ye has presented matOptimize at The Annual International Conference on Intelligent Systems for Molecular Biology (ISMB), held virtually on July 25-30, 2021. You can find his slides here.

CHAPTER

SIX

RIPPLES

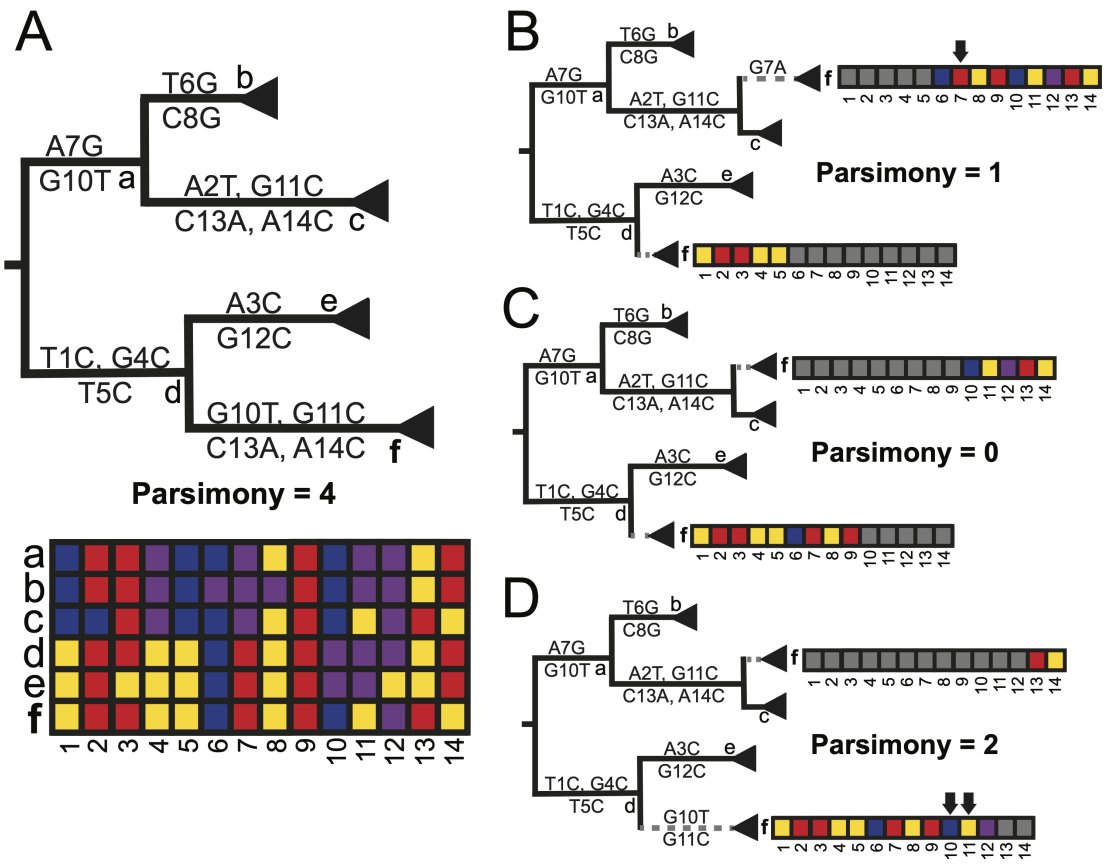
RIPPLES (Recombination Inference using Phylogenetic PLacEmentS) is a program used to detect recombination events in large mutation annotated tree (MAT) files.

6.1 Installation

To install RIPPLES, simply follow the directions for *installing UShER*, and RIPPLES will be included in your installation.

6.2 Methodology

RIPPLES is a program thatto rapidly and sensitively detect recombinant nodes and their ancestors in a mutation-annotated tree (MAT). RIPPLES exploits the fact that recombinant lineages arising from diverse genomes will often be found on "long branches" which result from accommodating the divergent evolutionary histories of the two parental haplotypes. Therefore, RIPPLES first identifies long branches in a MAT. RIPPLES then exhaustively breaks the potential recombinant sequence into distinct segments that are differentiated by mutations on the recombinant sequence and separated by up to two breakpoints. For each set of breakpoints, RIPPLES places each of its corresponding segments using maximum parsimony to find the two parental nodes – a donor and an acceptor – that result in the highest parsimony score improvement relative to the original placement on the global phylogeny. The nodes for which a set of breakpoints along with two parental nodes can be identified that provide a parsimony score improvement above a user-specified threshold are reported as recombinants. The figure below illustrates the RIPPLES algorithm.



In the partial phylogeny shown above, node **f** is the result of a recombination event between nodes **c** and **d**. In order to detect this, RIPPLES searches possible donor and acceptor pairs at which to place **f** for the set of placements resulting in greatest parsimony improvement. In panel B, the breakpoint is assumed to be after position 5 in the genome. Upon partially placing **f** as a descendant of node **d** and as a sibling to node **c** using this breakpoint, the parsimony score is 1, as the terminal branch for node **f** as a sibling of **c** contains one mutation (G7A). In panel C, however, an optimal donor, acceptor, and breakpoint are found. With the breakpoint after position 9 in the genome, this placement of **f** results in a parsimony score of 0. Given that the initial parsimony score of **f** was 4 (panel A), this is an overall parsimony score improvement of 4, indicating that this is likely a true recombination event.

6.3 Options

```
--input-mat (-i): Input mutation-annotated tree file [REQUIRED]. If only this argument␣
↪is set, print the count of samples and nodes in the tree.
--outdir (-d): Write all output files to the target directory. Default is current␣
↪directory.
--branch-length (-l): Minimum length of the branch to consider for recombination events.␣
↪Default = 3.
--min-coordinate-range (-r): Minimum range of the genomic coordinates of the mutations␣
↪on the recombinant branch. Default = 1,000.
--max-coordinate-range (-R): Maximum range of the genomic coordinates of the mutations␣
↪on the recombinant branch. Default = 10,000,000.
--samples-filename (-s): Restrict the search to the ancestors of the samples specified␣
```

(continues on next page)

(continued from previous page)

```
↪in the input file.
--parsimony-improvement (-p): Minimum improvement in parsimony score of the recombinant␣
↪sequences during the partial placement. Default = 3.
--num-descendants (-n): Minimum number of leaves that node should have to be considered␣
↪for recombinatino. Default = 10.
--threads (-T): Number of threads to use when possible. Default = use all available␣
↪cores.
--help (-h): Print help messages.
```

6.4 Usage

RIPPLES minimally requires an input MAT to search for recombination events. An example workflow is available *here*.

6.5 Publications

- Turakhia Y, Thornlow B, Hinrichs A, McBroome J, Ayala N, Ye C, De Maio N, Haussler D, Lanfear R, and Corbett-Detig R. Pandemic-Scale Phylogenomics Reveals Elevated Recombination Rates in the SARS-CoV-2 Spike Region., *bioRxiv*. 2021.

CHAPTER

SEVEN

BTE

7.1 Overview

BTE is a Cython API wrapper around the highly optimized phylogenetics library underlying the Pandemic Phylogenetics toolkit. It allows the user to leverage the power of the Mutation Annotated Tree file format and library in their Python scripts, allowing for efficient and effective analysis of global SARS-CoV-2 and other pathogen phylogenies. This package is generally intended as a replacement for ETE3, Biopython.Phylo, and similar Python phylogenetics packages for Mutation Annotated Trees (MATs). Using standard packages with MATs requires conversion to newick and the maintenance of mutation annotations as a separate data structure, generally causing inconvenience and slowing both development and runtime. BTE streamlines this process, allowing for efficient and convenient use of MATs in a Python development environment!

BTE can be found at its repository. This repository includes detailed installation instructions, a quickstart, and a dedicated documentation.

7.2 Installation

BTE is available via bioconda.

```
conda install -c bioconda bte
```

Local installation instructions can be found here.

7.3 Quickstart

Download the latest public SARS-CoV-2 tree:

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/public-latest.
↪all.masked.pb.gz
```

And proceed directly to your analysis in Python!

```
import bte
tree = bte.MATree("public-latest.all.masked.pb.gz")
```

Example analyses can be found at the BTE binder and its associated repository.

CHAPTER

EIGHT

STRAIN PHYLOGENETICS

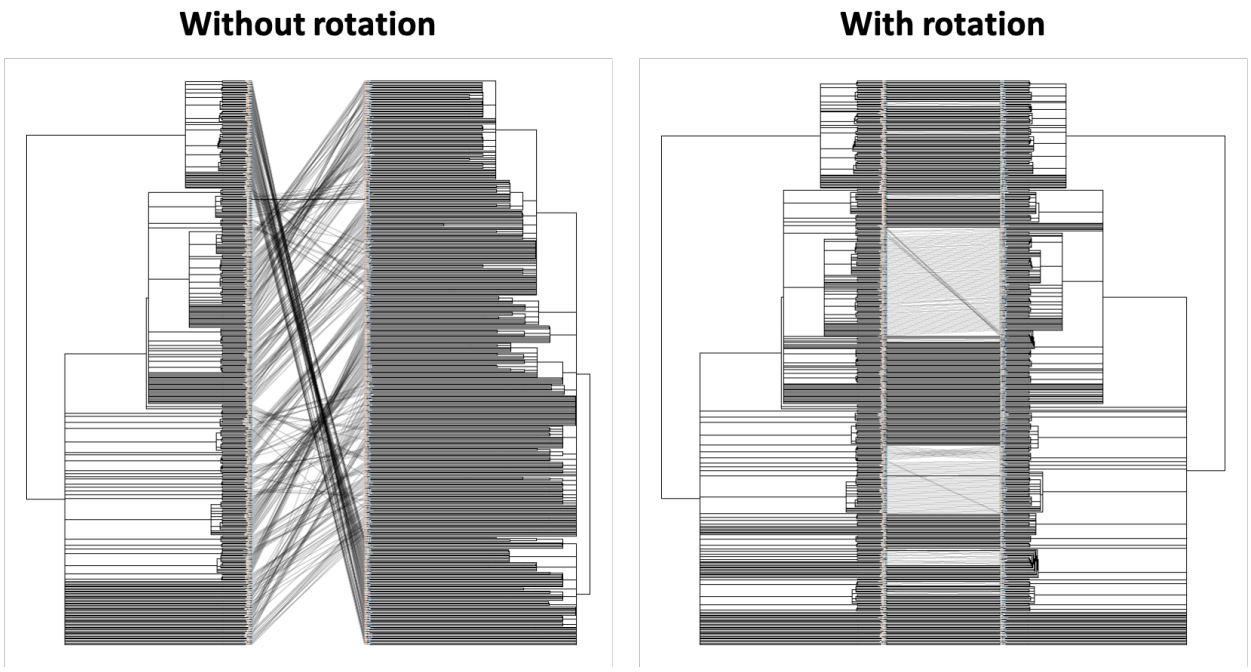
8.1 RotTrees

RotTrees enables quick inference of congruence of tanglegrams. This is particularly useful for SARS-CoV-2 phylogenomics due to multiple groups independently analyzing data-sets with many identical samples. Previous tanglegram visualization software, such as cophylo and Dendroscope3 rely on fewer rotations to minimize crossings over, which is inadequate for phylogenies on the scale of SARS-CoV-2. We implemented a quick heuristic to produce vastly improved tanglegrams.

First, ensure that `tree_1.nh` and `tree_2.nh` have identical sets of samples. Then, use as follows:

```
./build/rotate_trees --T1 tree_1.nh --T2 tree_2.nh --T1_out rot-tree_1.nh --T2_out rot-
↪tree_2.nh
```

The above command produces rotated trees (`rot-tree_1.nh` and `rot-tree_2.nh`) with a much improved tanglegram as seen below (images generated with the help of cophylo, setting rotate to FALSE).



Below is a GIF of approximately 20 frames showing various operations of the tree rotation algorithm operating on a much larger pair of trees (~4k leaves)

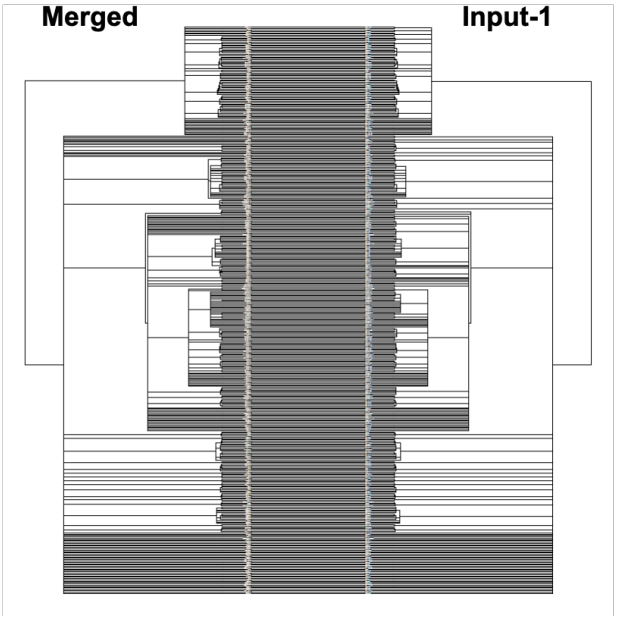
8.1.1 Options

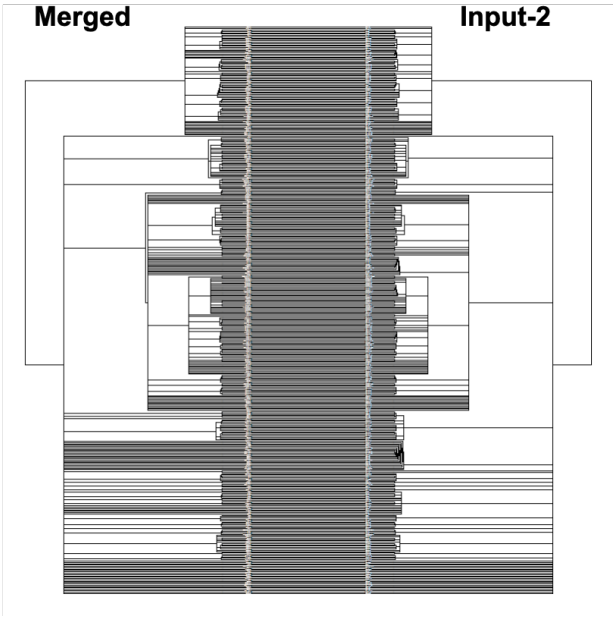
```
--T1: Input tree 1 (in Newick format).
--T2: Input tree 2 (in Newick format).
--T1_out: Output tree 1 (in Newick format).
--T2_out: Output tree 2 (in Newick format).
--max_iter: Maximum number of iterations. Default = 100
--help: Print help messages.
```

8.2 TreeMerge

```
python3 scripts/tree_merge.py -T1 tree_1.nh -T2 tree_2.nh -symmetric 1 -T_out symm-
↪merged-tree_1-tree_2.nh
```

The above command produces a merged tree (`symm-merged-tree_1-tree_2.nh`) from two input trees (`tree_1.nh` and `tree_2.nh`) that is maximally resolved and compatible with both input trees (refer to our manuscript for more details). Below are the resulting tanglegrams of the resulting merged tree with the two input trees (after applying tree rotation). The above command can also be used without the symmetric flag for its asymmetric version (where the first input tree is given a priority to resolve the merged tree) or using the intersectOnly flag that produces a simple consensus of the two input trees.





8.2.1 Options

```
-T1: tree 1 (in Newick format). (REQUIRED)
-T2: tree 2 (in Newick format). (REQUIRED)
-T-out: output tree filename (in Newick format). (REQUIRED)
-intersectOnly: output intersection (instead of a maximal merge) of T1 and T2.
-symmetric: output symmetric merge of T1 and T2.
--help (-h): Print help messages.
```

8.3 Find parsimonious assignments

```
./build/find_parsimonious_assignments --tree tree/pruned-sumtree-for-cog.nh --vcf vcf/
↪tree_1.vcf > tree_1_PARSIMONY.txt
```

The above command reads the tree topology of the input Newick file and assigns an internal numeric label for each internal node (ignoring the internal labels and branch lengths if already provided by the input Newick). The first two lines of the output file print the input tree with internal nodes labelled in Newick format. The output is too large to display, so we view the first 1000 characters using the command below.

```
head -c 1000 tree_1_PARSIMONY.txt
```

For each variant/site in the VCF file, the output file then displays the allele frequency for each alternate variant, its total parsimony score, the list of nodes (comma-separated, if its length is <=4) for which the branches leading to it have acquired a mutation (forward [F] or backward [B], the sizes of the clades affected by those mutations and a list of flagged leaves which are affected by a mutation affecting 3 or fewer leaves.

8.3.1 Options

```
--tree: Input tree file.
--vcf: Input VCF file (in uncompressed or gzip-compressed format).
--threads: Number of threads. Default = 40
--print-vcf: Print VCF with variants resolved instead of printing a parsimony file.
--help: Print help messages.
```

8.4 Identify extremal sites

```
python3 scripts/identify_extremal_sites.py -in tree_1_PARSIMONY.txt
```

The above command can be used for identifying and flagging extremal sites i.e. sites having exceptional parsimony scores relative to their allele frequencies and therefore also suspected to contain systematic errors. The above command identifies 6 extremal sites (C11074T, C27046T, T13402G, A3778G, G24390C, G26144T) with a phylogenetic instability value of 3.03. For the precise definition of extremal sites and phylogenetic instability, refer to our manuscript referenced at the bottom. The code also provides an ability to ignore high-frequency C>T and G>T mutations using optional flags.

```
python3 scripts/identify_extremal_sites.py -in tree_1_PARSIMONY.txt -ignoreCtoT=1 -
↪ignoreGtoT=1
```

The above command identifies three extremal sites (T13402G, A3778G, G24390C) with a phylogenetic instability value of 2.32. To create a figure requires installing R and the plyr package.

8.4.1 Options

```
-in: Input parsimony file.
-ignoreCtoT: Set to 1 to ignore C>T sites (default=0)
-ignoreGtoT: Set to 1 to ignore G>T sites (default=0)
--help (-h): Print help messages.
```

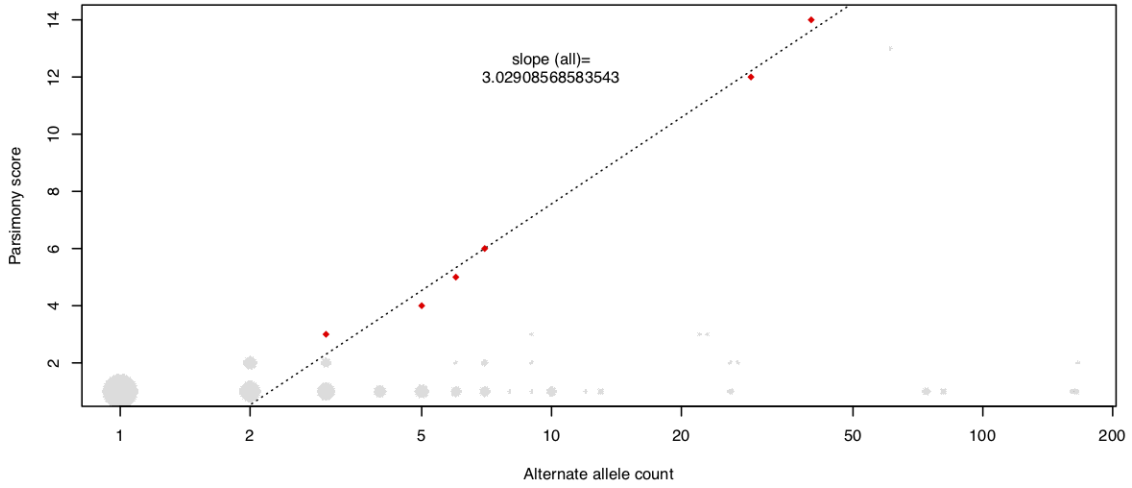
8.5 Plot extremal sites

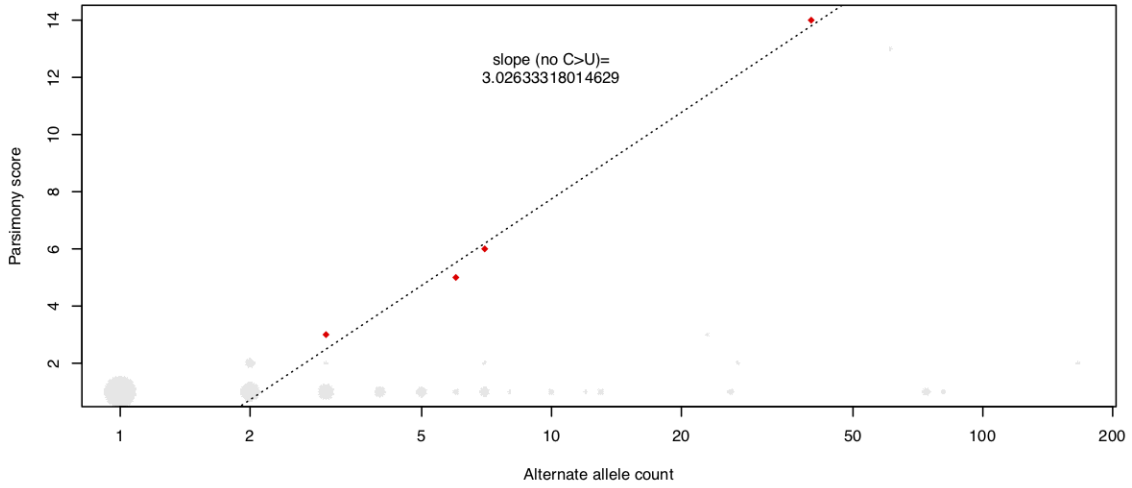
```
python3 scripts/generate_plot_extremal_sites_data.py -in tree_1_PARSIMONY.txt > plot_
↪extremal_sites_data.txt
```

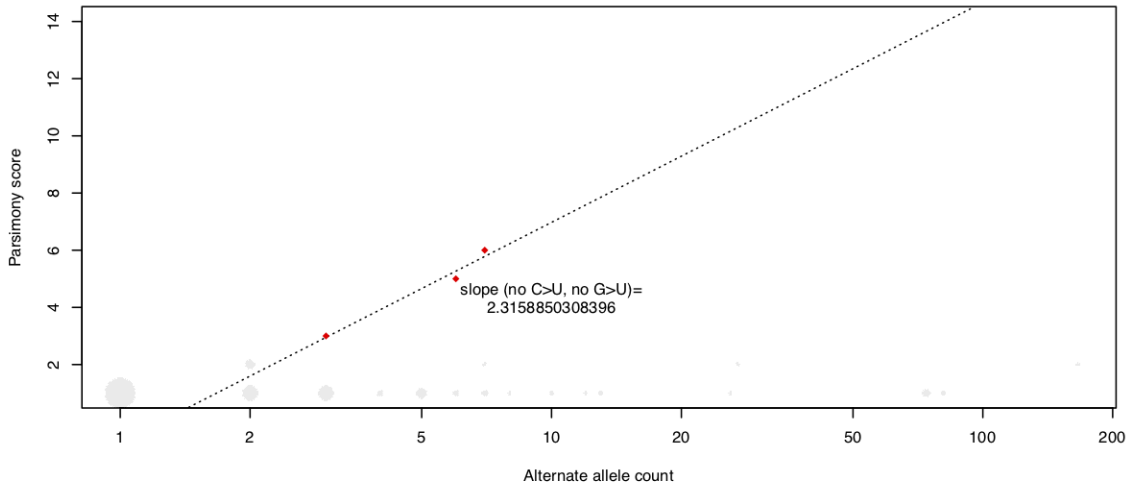
The above commands create raw input data for the extremal sites plot.

```
Rscript --vanilla scripts/plot_parsimony.r plot_extremal_sites_data.txt extremal_sites_
↪plot.pdf
```

Next, the R command accepts the generated data and creates a log(allele count) by parsimony plot for all variant sites in a given vcf. It produces three plots, one of all data, one ignoring C>U mutations and one ignoring C>U and G>U mutations, as shown below.


slope (all)=
3.02908568583543
Parsimony score
Alternate allele count


slope (no C>U)=
3.02633318014629
Parsimony score
Alternate allele count


slope (no C>U, no G>U)=
2.3158850308396
Parsimony score
Alternate allele count

8.6 Presentations

We have presented this package and analyses on GISAID data at the Covid-19 Dynamics & Evolution Meeting, held virtually on October 19-20, 2020. You can find our slides here.

8.7 Publications

- Turakhia Y, De Maio N, Thornlow B, Gozashti L, Lanfear R, Walker C, Hinrichs A, Fernandes J, Borges R, Slodkowicz G, Weilguny L, Haussler D, Goldman N, and Corbett-Detig R. Stability of SARS-CoV-2 Phylogenies. *PLOS Genetics*. 2020. 16(11): e1009175.
- De Maio N, Walker C, Turakhia Y, Lanfear R, Corbett-Detig R, and Goldman N. Mutation rates and selection on synonymous mutations in SARS-CoV-2. *bioRxiv*. 2020.
- DeMaio N, Walker C, Borges R, Weilguny L, Slodkowicz G, and Goldman N. Issues with SARS-CoV-2 sequencing data. *Virological*. 2020.
- Gozashti L, Walker C, Goldman N, Corbett-Detig R, and DeMaio N. Updated analysis with data from 13th November 2020. *Virological*. 2020.

CHAPTER

NINE

TUTORIALS

This document contains example workflows for UShER, matUtils, and RIPPLES.

9.1 Using RIPPLES to detect recombination in new sequences

Users interested in recombination may want to use RIPPLES to search for recombination events in their set of samples. In this example, we will search for recombination events involving a set of samples based on this preprint by Jackson et al..

First, download the latest public tree, a set of sample sequences to search for recombination, the reference genome, and sites to mask:

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/public-latest.
↪all.masked.pb.gz
wget https://raw.githubusercontent.com/bpt26/usher_wiki/main/docs/source/test_samples.fa
wget https://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/bigZips/wuhCor1.fa.gz
wget https://raw.githubusercontent.com/W-L/ProblematicSites_SARS-CoV2/master/problematic_
↪sites_sarsCov2.vcf
```

Note: Your exact results may be slightly different than what is shown here, as the public tree is updated daily. To get the same tree used in this tutorial, use the following command:

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/2021/08/04/
↪public-2021-08-04.all.masked.pb
```

Then, mask problematic sites using the downloaded .vcf:

```
gunzip wuhCor1.fa.gz
mafft --thread 10 --auto --keeplength --addfragments test_samples.fa wuhCor1.fa >␣
↪aligned_seqs.fa
faToVcf -maskSites=problematic_sites_sarsCov2.vcf aligned_seqs.fa aligned_seqs.vcf
```

Then, use UShER to add your samples to the protobuf:

```
usher -T 10 -i public-latest.all.masked.pb.gz -v aligned_seqs.vcf -o user_seqs.pb
```

Then, use RIPPLES to search for recobination events involving these samples in the tree:

```
mkdir USER_SAMPLES/
grep -e '>' test_samples.fa | perl -pi -e 's/>//' > user_samples.txt
ripples -i user_seqs.pb -s user_samples.txt -d USER_SAMPLES/ -T 10
```

After a few minutes, RIPPLES produces files *recombination.tsv* and *descendants.tsv*. The last two columns in the *recombination.tsv* file represent the parsimony score improvement. Suppose that we are interested specifically in recombinant nodes with highest parsimony score improvement, as these are more likely to reflect true recombination events. This command yields all recombination events with parsimony improvements greater than 7:

```
awk '$11 - $12 > 7' USER_SAMPLES/recombination.tsv
```

The results should look something like this:

```
node_49072  (0,241) (10870,11288) node_111282 n 41  node_49068  n 12  12  12  4
node_49072  (0,241) (14408,14676) node_179776 n 36  node_49068  n 12  12  12  4
node_49072  (0,241) (7728,10870)  node_179776 n 36  node_49068  n 12  12  12  3
node_49072  (0,241) (6954,7728) node_146650 n 40  node_49068  n 12  12  12  4
node_49072  (241,913) (7728,10870)  node_179776 n 36  node_49068  n 12  12  12  4
node_49072  (7728,10870)  (29218,GENOME_SIZE) node_49068  n 12  node_179776 n 36  12  12␣
↪ 4
node_92879  (0,445) (21255,22227) node_48600  n 28  node_151816 n 20  10  10  1
node_92879  (0,445) (20410,22227) node_48600  n 28  node_151816 n 20  10  10  2
node_92900  (0,241) (23403,23604) node_253479 n 26  node_136388 n 25  13  13  5
node_92900  (0,241) (23271,23604) node_253479 n 26  node_136388 n 25  13  13  5
node_92900  (0,241) (23063,23271) node_253479 n 26  node_136388 n 25  13  13  4
node_92900  (0,241) (22444,23063) node_253479 n 26  node_98962  n 23  13  13  3
node_92900  (0,241) (21123,23063) node_253479 n 26  node_98962  n 23  13  13  4
node_92900  (0,241) (18877,21123) node_253479 n 26  node_108815 n 25  13  13  4
node_92900  (241,1947)  (21123,23271) node_253479 n 26  node_119821 n 22  13  13  5
node_92900  (241,1947)  (22444,23063) node_253479 n 26  node_119821 n 22  13  13  4
node_92900  (241,1947)  (18877,21123) node_253479 n 26  node_108815 n 25  13  13  5
node_92900  (1947,2319) (22444,23063) node_253479 n 26  node_119821 n 22  13  13  5
node_92900  (18131,21123) (28977,29742) node_108815 n 25  node_253479 n 26  13  13  5
node_92900  (21779,22444) (28977,29742) node_98068  n 27  node_253479 n 26  13  13  5
node_92900  (22444,23063) (28977,29742) node_100243 n 23  node_253479 n 26  13  13  5
```

We can then parse the *descendants.tsv* file to look at the descendants for each of these nodes:

```
awk '$1 == "node_49072"' USER_SAMPLES/descendants.tsv
awk '$1 == "node_92879"' USER_SAMPLES/descendants.tsv
awk '$1 == "node_92900"' USER_SAMPLES/descendants.tsv
```

These commands yield the descendants for the nodes of interest. We find that node_49072 is an ancestor of the sample s1. node_92879 is an ancestor of the samples s4, s5, and s6. node_92900 is an ancestor of samples s8, s9, and s10.

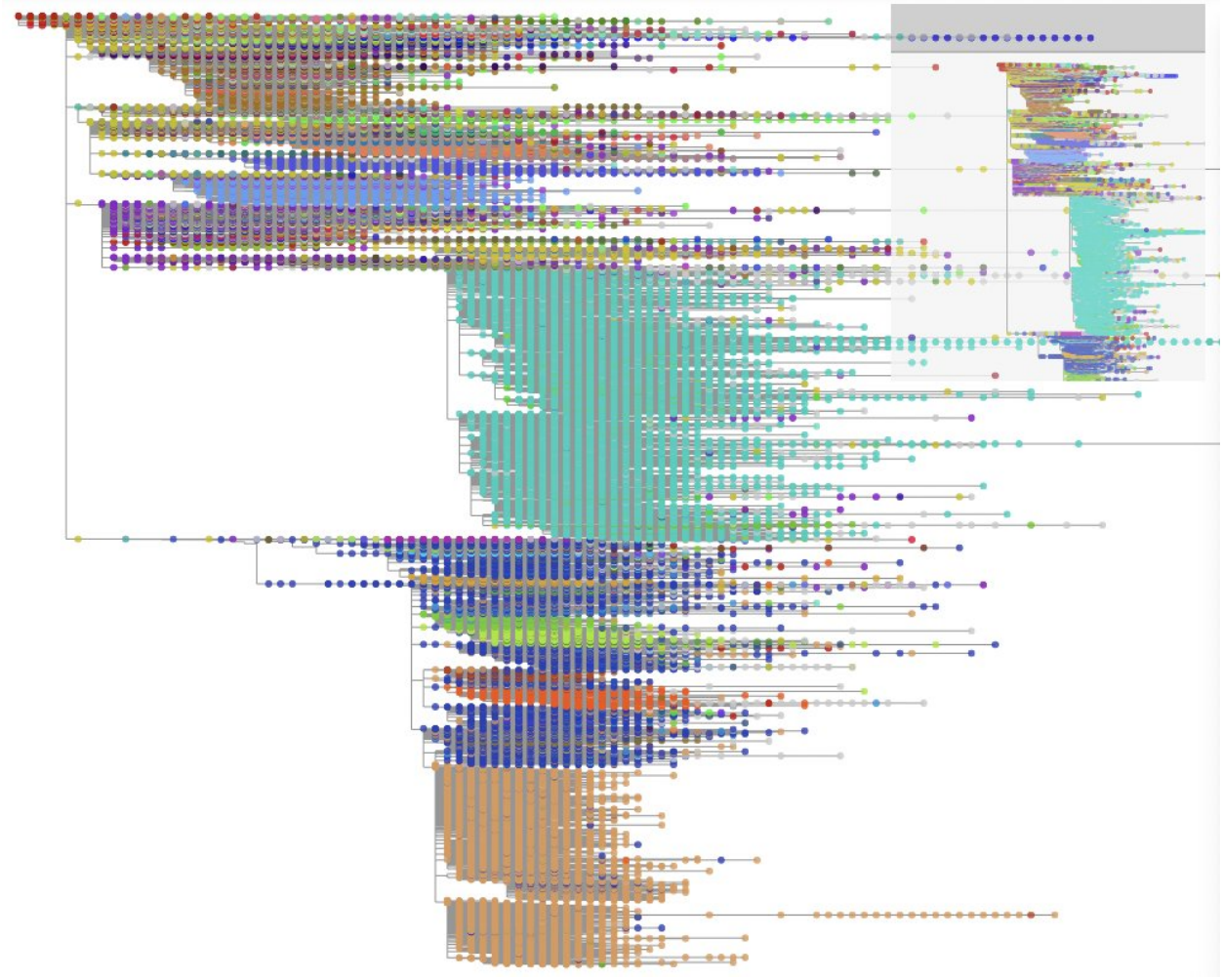
9.1.1 Snakemake Workflow

Alternatively, after installing snakemake, the following set of commands may be used to search for evidence of recombination in a user-input set of sequences:

```
cd usher/workflows
snakemake --use-conda --cores 4 --config FASTA="/path/to/fasta" RUNTYPE="ripples"
```

9.2 Using Taxonium to visualize phylogenies

We recommend using Taxonium to visualize these trees. An example of a phylogeny visualized by this software is shown below:



You can use Taxoniumtools to convert from UShER to Taxonium format.

```
conda install -c conda-forge python
pip install taxoniumtools
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/public-latest.
↪metadata.tsv.gz && gunzip public-latest.metadata.tsv.gz
wget http://hgdownload.soe.ucsc.edu/goldenPath/wuhCor1/UShER_SARS-CoV-2/public-latest.
```

(continues on next page)

(continued from previous page)

```
↪all.masked.pb.gz
wget https://raw.githubusercontent.com/theosanderson/taxonium/master/taxoniumtools/test_
↪data/hu1.gb
usher_to_taxonium --input public-latest.all.masked.pb.gz --output public-latest-taxonium.
↪jsonl.gz --metadata public-latest.metadata.tsv --genbank hu1.gb --columns genbank_
↪accession,country,date,pangolin_lineage
```

You then open that *.jsonl.gz* file on Taxonium directly, or with the Taxonium desktop app which may handle large trees better

9.2.1 Snakemake Workflow

For simplicity, we include the above set of commands as a snakemake workflow. Here, the user-specified input is a set of sequences to be added to the latest public tree. This workflow can be run using the following commands:

```
cd usher/workflows
snakemake --use-conda --cores 4 --config FASTA="path/to/fa/file" RUNTYPE="taxodium"
```

9.3 Basic matUtils Workflow

Though it is bundled and installed alongside UShER, matUtils is more than an output processor for UShER commands. It can be used in independent workflows that begin with one of our publicly-provided MAT protobuf files, or an Augur-formatted MAT JSON file as used by Nextstrain. matUtils can be used to explore large trees in deep detail, parsing and manipulating trees of a size few other tools can manage efficiently. The first step to using one of these public files is *matUtils summary*, which can calculate basic statistics or summarize sample, clade, or mutation-level frequency information.

```
matUtils summary -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz
```

This particular tree has 757500 unique samples represented in it. We can further explore this dataset with another *matUtils summary* command:

```
matUtils summary -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz -A -d summary_
↪out
```

Let's say we're interested in recurrent, or homoplasic, mutations across this tree. The mutations summary file contains the number of independent occurrences of a mutation across the tree.

```
awk '$2 >= 500' summary_out/mutations.tsv
```

C>T mutations are very overrepresented in this set, being a common mutation and error type. We're concerned about correctly identifying real homoplasic mutations, instead of coincident or erroneous mutations, so we filter the dataset down to higher-quality placements and samples.

```
matUtils extract -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz -a 3 -b 5 -o␣
↪filtered.pb
matUtils summary -i filtered.pb
```

After filtering, our tree contains 701375 samples, which is 92% of the original tree size. Let's see how our homoplasic mutation output looks.

```
matUtils summary -i filtered.pb -m filtered_mutations.tsv
awk '$2 > 500' filtered_mutations.tsv
```

By filtering 8% of our tree, we have removed most of the mutations with more than five hundred unique occurrences. The C>T bias is still present, but we can more comfortably proceed to analyze these mutations. The most homoplasic mutation, a significant outlier with more than a thousand occurrences after filtering, is G7328T. This is a mutation in ORF1A, part of the replicase protein which the virus uses to duplicate itself in the host, which causes an amino acid change from alanine to serine at position 2355.

If we were interested in following up on this potential homoplasy, we have a few options. We may want to generate a new protobuf file containing only samples with this specific mutation, along with a JSON for visualization and additional sample path information. We can perform all these operations with a single command.

```
matUtils extract -i filtered.pb -m G7328T -o G7328T.pb -j G7328T.json -S G7328T_sample_
↪paths.txt
```

The expected output can be viewed here or you can upload your generated JSON at Auspice.

In this view, we can choose to highlight each branch and node by whether they contained our query mutation. We can see that the majority of occurrences of G7328T are single nodes- having just occurred- and that the majority are from the USA, though from all across the phylogenetic tree. Further analysis would be required to validate or interpet these results, but this procedure clearly demonstrates the potential for matUtils for rapid exploratory analysis using large public datasets.

9.3.1 Snakemake Workflow

We also provide a snakemake workflow to add user-input sequences in .fasta format to the latest public tree, and extract subtrees of size 500 for each. These subtrees are output in .json format for simple drag-and-drop visualization with Auspice. This workflow can be run using the following commands:

```
cd usher/workflows
snakemake --use-conda --cores 4 --config FASTA="path/to/fa/file" RUNTYPE="matUtils"
```

9.4 Example Uncertainty Workflow

In this example we will calculate uncertainty metrics for samples belonging to clade B.1.500 and visualize them on auspice.

Download the example protobuf file public-2021-05-17.all.masked.nextclade.pangolin.pb.gz (protobuf file containing the mutation annotated tree with clade annotations)

The first step is generating a visualizable JSON of the clade of interest, along with getting the names of samples involved. This is done with matUtils extract. In our example, we will get the samples associated with a small pangolin clade.

```
matUtils extract -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz -c B.1.500 -u␣
↪b1500_samples.txt -j b1500_viz.json
```

The second step is to call matUtils uncertainty. The input PB is the original PB, with the sample selection text file, instead of a subtree pb generated with -o. This is because its going to search for placements all along the original tree; if a subtree .pb was passed, it would only search for placements within that subtree.

```
matUtils uncertainty -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz -s b1500_
↪samples.txt -e b1500_uncertainty.tsv -o b1500_placements.tsv
```

These can now be uploaded for visualization by drag and drop onto the auspice website. Drag and drop the b1500_viz.json first, then the tsv files second. Alternatively, one of the metadata files can be included in JSON generation by matUtils extract.

```
matUtils extract -i public-2021-05-17.all.masked.nextclade.pangolin.pb.gz -s b1500_
↪placements.txt -z 100 -M b1500_placements.tsv -j b1500_annotated.json
```

The placement metadata file in this example is also passed into -s as well as -M to ensure that all samples with placement information are included in the output. -z fills out additional random samples to contextualize the data. b1500_uncertainty.tsv can be drag-and-dropped onto the Auspice view to display additional colorings.

The expected output can be viewed in full here.

9.5 Example Introduce Workflow

Note: This feature is currently under active development! We are actively soliciting feedback on the usefulness of the current implementation, additional features that would be valuable, or directions to take this type of analysis. Please reach out on the GitHub or directly to me via my email, jmcbroom@ucsc.edu.

In this example we will infer and investigate introductions of SARS-CoV-2 into Spain using public information on the command line and visualize an example introduction of interest with Auspice.

Before beginning, download the example protobuf file public-2021-04-20.all.masked.nextclade.pangolin.pb

We need a region to analyze; in this example, we are going to use Spain, as it has a few hundred associated samples in the public data and is a solid representative example. We need to generate the two-column tab-separated file we use as input to *matUtils introduce*.

```
matUtils summary -i public-2021-04-20.all.masked.nextclade.pangolin.pb -s 420_sample_
↪parsimony.txt
grep "Spain" 420_sample_parsimony.txt | awk '{print $1"\tSpain"}' > spanish_samples.txt
```

We can now apply *matUtils introduce* using this file as input.

```
matUtils introduce -i public-2021-04-20.all.masked.nextclade.pangolin.pb -s spanish_
↪samples.txt -o spanish_introductions.txt
```

The output table (spanish_introductions.txt) has columns for the sample, the identifier of the introduction node, the confidence of that introduction point being in region, the confidence of the parent of that introduction point being in region, the number of mutations between the sample and this introduction point, any clades associated with the introduction point, and the path of mutations to the point of introduction.

Generally the confidence of the introduction point will be greater than 0.5 and the confidence of the parent of the introduction point will be less than 0.5, marking the point on the history where we stop being confident that the represented ancestral sequence was local to the region.

We can count the number of unique introductions into our region of interest- in this case Spain- using awk.

```
awk '{print $2}' spanish_introductions.tsv | sort | uniq -c | sort -r | head -25
```

We find 216 unique introductions into Spain, of which 175 are associated with only a single sample, from 295 total samples. This may suggest that Spain has a lot of movement in and out of the country, or that sampling is biased towards travelers. It may also simply reflect that Spain is undersampled and the relative number of introductions is high enough that most new regional clades are sampled only once or not at all.

There are some interesting cases of clades from a single introduction, however. The clade introduced at the internal node "96055" contains 9 closely related samples from Spain and are all members of the variant of concern B.1.1.7.

```
awk '$2 == "96055"' spanish_introductions.tsv
```

Warning: Internal node names are not maintained in the protobuf and are not guaranteed to be consistent between protobufs with differing content. The path of mutations to the point of introduction will generally be consistent, however.

The first entry of this output is reproduced here, sans the mutation path.

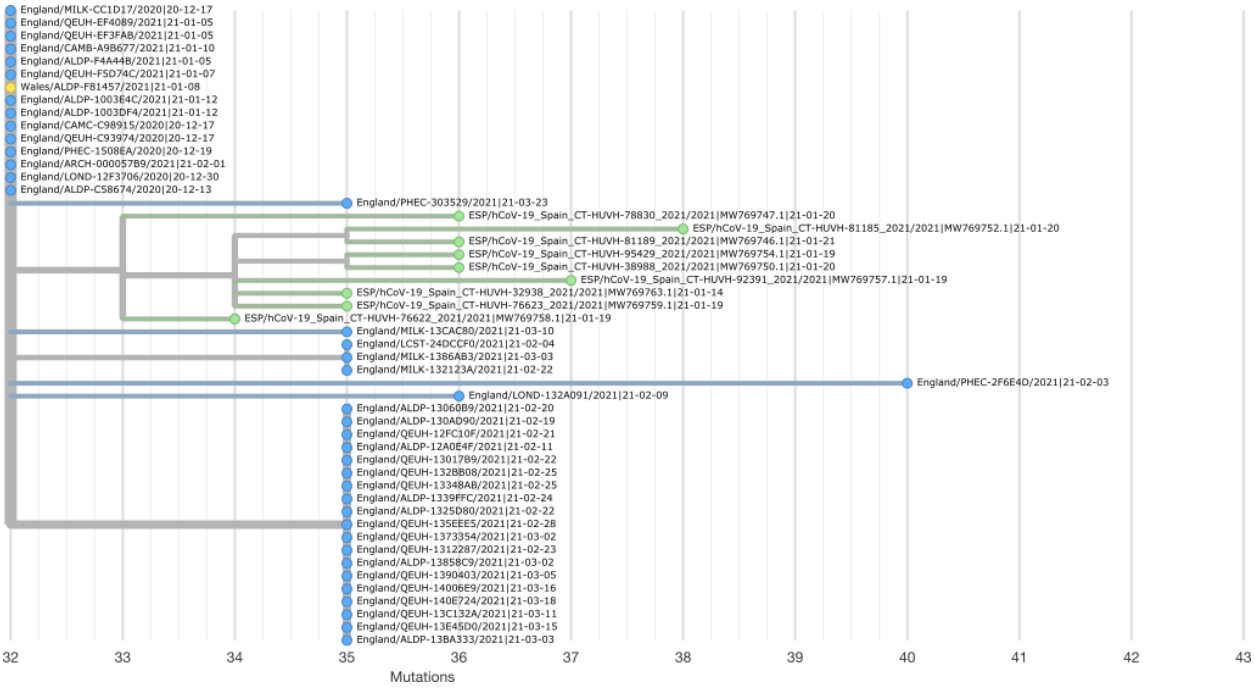
```
ESP/hCoV-19_Spain_CT-HUVH-32938_2021/2021|MW769763.1|21-01-14        96055   2       4.5
↪    2021-Jan-14     2021-Jan-21     9       51              1       0.0431655       3
```

We can see that this introduction point is very confidently in Spain (confidence of 1 in column 9, as every descendent is from Spain) but that the parent of that introduction point is very confidently NOT from Spain (confidence of 0.043 to be in Spain). This makes this a strongly supported introduction of a variant of concern into our region. Samples in this cluster span the week from the 14th to the 21st of January 2021, meaning that it has a cluster growth score of 9/(1+1) or 4.5, which puts it second on our ranked cluster list. Let's take a closer look by visualizing it on the Auspice web interface.

To do this, first we will need to generate an auspice-compatible JSON containing our introduction set and some context samples. We can do this by selecting one of our samples and extracting the context to a JSON with *matUtils extract*.

```
matUtils extract -i public-2021-04-20.all.masked.nextclade.pangolin.pb -k "ESP/hCoV-19_
↪Spain_CT-HUVH-76622_2021/2021|MW769758.1|21-01-19:50" -j spanish_introduction.json
```

This JSON can be drag-and-dropped onto the Auspice web interface. You can view the expected output interactively here.



Samples from England appear in blue, Spain in green, and Wales in yellow in this image. We can see that our group of 9 B.1.1.7 samples forms a clear clade that was likely introduced into Spain from England.

Additional steps we could include are the generation of metadata tsv/csv for Auspice, the inclusion of more regions, and the inclusion of phylogeographic statistics with -a on our call to *matUtils introduce*. The latter increases the runtime of the introduce command from a few seconds to about two minutes in this case.

Spain has an overall association index of 10.4 under a 95% confidence interval of (28.95,40.19) for the null that samples from this region are not phylogenetically associated. This is a very, very significant association score, which is normal for geographic regions, as naturally samples from the same region are more closely related to one another. The largest monophyletic clade size is 9, representing our specific introduction of interest.

Our specific introduction of interest itself also has a monophyletic clade size of 9 (being pure with 9 samples) and an association index of 0, representing that it is purely in-region and is maximally associated.

Including additional public region information in the input two-column tsv would also allow us to explore potential origins of each introduction, such as how England appears to be the origin in our example, or estimate relative levels of migration to and from Spain to other countries across the world. Origin and migration must be interpreted cautiously, however, due to extensive sampling bias by country (England and the UK contribute a large part of publicly available sequence information, and are therefore more likely to be identified as the origin of an introduction, et cetera).

9.5.1 Snakemake Workflow

We also provide a snakemake workflow to search for unique introductions within a user-input set of samples. This workflow can be run using the following commands:

```
cd usher/workflows
snakemake --use-conda --cores 4 --config FASTA="/path/to/fasta" RUNTYPE="introduce"
```

9.6 Calculating by-mutation RoHo with *matUtils summary* and Python

The Ratio of Homoplasic Offspring (RoHo) is the log10 of the ratio of the descendents of a clade with a specific mutation to a sister clade without a mutation. It is negative when the clade with the mutation is smaller than its sister, and positive when it is larger.

As originally formulated, this metric assumes a bifurcating, fully resolved tree. As the MAT is not a resolved tree and contains many internal polytomies, we use the median size of all sister clades for calculation. The number of sister clades for each occurrence is included in the output table. The output table of summary is by-occurrence, so distributions of values associated with a specific homoplasic mutation can be selected by finding all rows with that mutation in the first column.

Before beginning, download the example protobuf file public-2021-06-09.all.masked.nextclade.pangolin.pb.gz

```
matUtils summary -i public-2021-06-09.all.masked.nextclade.pangolin.pb.gz -R roho_scores.
↪tsv
```

The resulting table contains the following columns:

```
mutation    parent_node     child_count     occurrence_node offspring_with  median_
↪offspring_without        single_roho
```

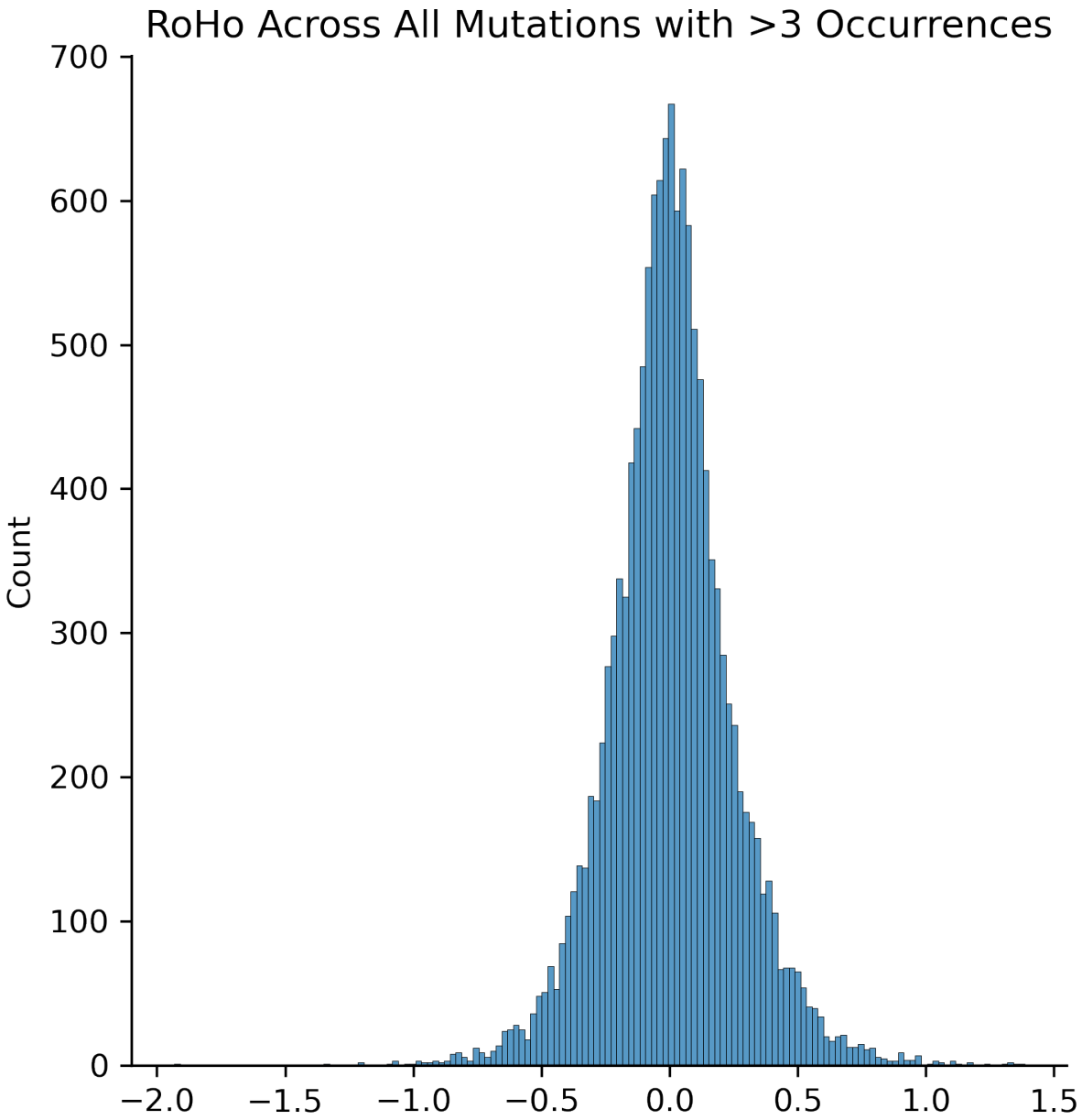
Mutation is the encoding string for a given nucleotide mutation (ReferenceLocationAlternative, e.g. A235G). The parent_node is the identifier of the parent of the node on which this mutation occurred and the occurrence_node is the identifier of that node proper; these numbers are used for downstream QC and analysis with matUtils extract. Child_count is the number of sister clades associated with the parent of the clade of interest. Offspring_with is simply the number of descendents from the clade with this mutation. Median_offspring_without is the median value of the

number of descendents of each sister clade to this clade (they all share the same parent_node). Single_roho is the log10 of offspring_with divided by median_offspring_without.

Using the standard data analysis Python packages Pandas and Seaborn, we can calculate and visualize the genome-wide distribution of RoHo values, specifically the mean RoHo value belonging to each homoplasic mutation with >3 occurrences.

```
import pandas as pd
import seaborn as sns
import numpy as np
rdf = pd.read_csv('roho_scores.tsv',sep='\t')
by_mut_rohos = []
for m, sdf in rdf.groupby("mutation"):
    if sdf.shape[0] >= 3:
        by_mut_rohos.append(np.mean(sdf.single_roho))
print(np.mean(by_mut_rohos))
sns.displot(by_mut_rohos)
```

The mean RoHo of this distribution is 0.005, suggesting very little bias, though the negative tail is longer than the positive tail. The resulting plot is replicated below.



9.7 Example Amino Acid Translation Workflow

As mutations from the reference accumulate in sequenced SARS-CoV-2 samples, there may be multiple nucleotide mutations in a single codon. In this case, protein sequences computed using nucleotide mutations at leaves may be incorrect if another mutation in the same codon occured higher up the tree. `matUtils summary --translate` provides a way to compute the correct amino acid translations at each node.

In this workflow, we will output amino acid mutations at each node of a protobuf, and prepare an annotated JSON suitable for visualization with Auspice.

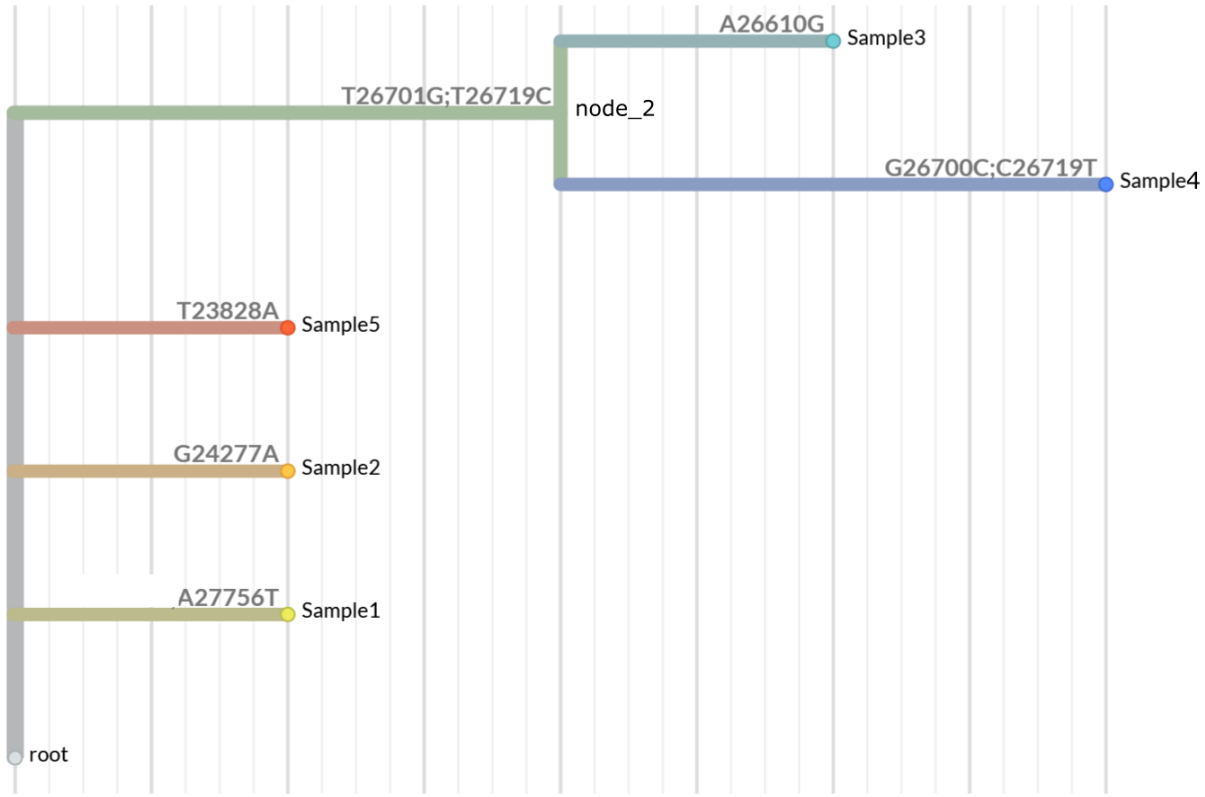
Prepare data

Download the example protobuf file (`translate_example.pb`), GTF file (`ncbiGenes.gtf`), and reference FASTA

(NC_045512v2.fa).

Note: The input GTF must follow the conventions specified here. If multiple CDS features are associated with a single gene_id, they must be ordered by start position.

The example protobuf file is a simple tree with the following structure:



Run matUtils summary to get amino acid translations

To call amino acid mutations in the mutation-annotated tree, run the following commmand:

```
matUtils summary --translate coding_mutations.tsv -i translate_example.pb -g ncbiGenes.
↪gtf -f NC_045512v2.fa
```

This will produce a TSV file named coding_mutations.tsv. The contents of the file are shown below

```
(line 1): node_id     aa_mutations            nt_mutations        leaves_sharing_
↪mutations
(line 2): Sample1     ORF7a:E121D;ORF7b:M1L   A27756T;A27756T     1
(line 3): Sample2     S:R905R                 G24277A             1
(line 4): Sample5     S:Y756N                 T23828A             1
(line 5): node_2      M:V60G;M:V66A           T26701G;T26719C     2
(line 6): Sample4     M:G60R;M:A66V           G26700C;C26719T     1
(line 7): Sample3     M:T30A                  A26610G             1
```

TSV Output format

Each line in the file corresponds to a node in the tree. Only nodes with mutations (including synonymous) are included.

aa_mutations are always delimited by a ; character, and can be matched with their corresponding nucleotide mutations in the nt_mutations column (also delimited by ;).

If there are mulitple nucleotide mutations in one node affecting a single codon (rare), they will be separated by commas in the nt_mutations column.

In the case that a single nucleotide mutation affects multiple codons, the affected codons are listed sequentially, and the nucleotide mutation is repeated in the nt_mutation column. An example of this case is shown in line 2 of the file above:

```
Sample1     ORF7a:E121D;ORF7b:M1L   A27756T;A27756T     1
```

A27756T mutates both the last codon of Orf7a and the start codon of Orf7b.

leaves_sharing_mutations indicates the number of descendant leaves of the node that share its set of mutations (including itself, if the node is a leaf).

Synonymous mutations

Synonymous mutations are included in the output. See the example in line 3 of coding_mutations.tsv:

```
Sample2     S:R905R                 G24277A             1
```

Mutation accumulation / Back-mutations

The following two lines demonstrate how matUtils summary --translate considers mutations higher in the tree when computing protein changes.

```
node_2      M:V60G;M:V66A           T26701G;T26719C     2
Sample4     M:G60R;M:A66V           G26700C;C26719T     1
```

Sample4 is a child of the internal node node_2. The mutation G26700C in Sample4 is in the same codon (M:60) as T26701G. The protein follows the path V (root) -> G (node_2) -> R (Sample4).

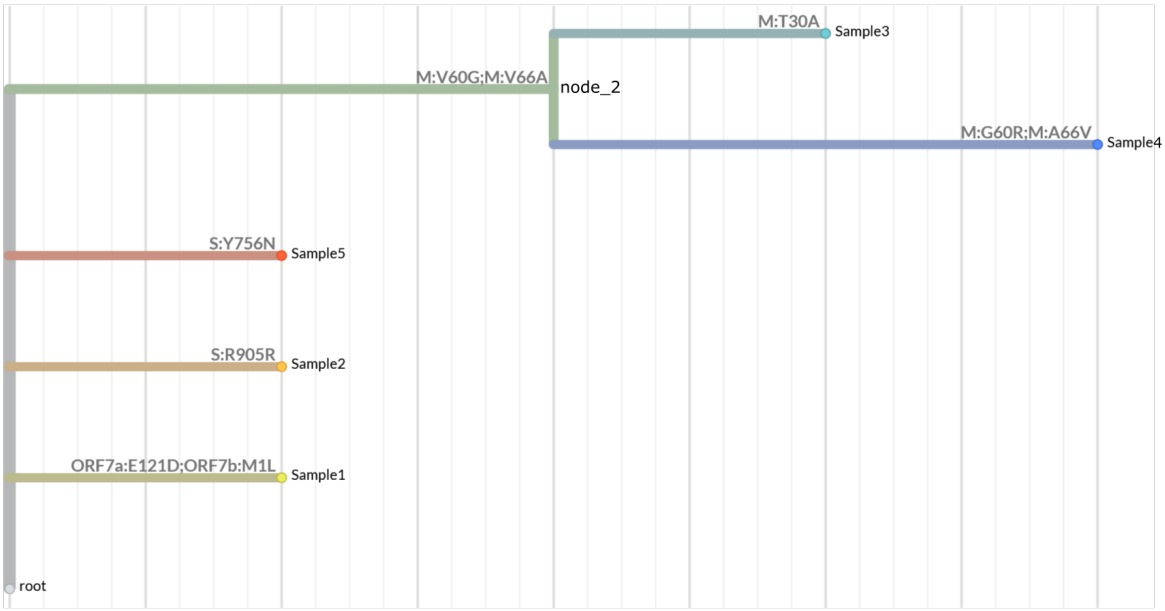
The above two lines also show an example of a back-mutation. The nucleotide mutation T26719C yields M:V66A in node_2, which is then back-mutated to V by C26719T in Sample4.

Run matUtils extract to annotate a JSON tree for visualization

To produce a JSON file with the metadata we produced above, run the following command:

```
matUtils extract -i translate_example.pb -M coding_mutations.tsv -j aa_annotated.json
```

The resulting JSON file can now be loaded into Nextstrain / Auspice. Here is the tree with amino acid annotations:



View the expected JSON in Nextstrain here.

You can use the `Branch Labels` menu in the sidebar to view the annotations.

9.7.1 Snakemake Workflow

We also include a snakemake workflow to translate mutations in a user-input .fasta file to amino acid substitutions in a lineage-aware manner. This workflow can be run using the following commands:

```
cd usher/workflows
snakemake --use-conda --cores 4 --config FASTA="/path/to/fa" RUNTYPE="translate"
```

9.8 Interacting Directly with Protobuf Files in Python [ADVANCED USERS]

Advanced users may desire to interface directly with the protobuf. The following is a brief tutorial on doing so. Google's general tutorial on interacting with protobuf in python can be found here. The instructions here can be applied to a number of additional languages supported by google as well, such as java, PHP, and ruby.

Note that this tutorial is specifically for directly manipulating the MAT protobuf file itself in Python; loading a MAT data structure into memory in Python and manipulating that structure is the province of the Cython extension BTE.

The first step is to call the protoc compiler to retrieve a MAT protobuf parser. Navigate to your Usher installation (or clone the github if you installed via conda) and call:

```
protoc -I=./ --python_out=./ ./parsimony.proto
```

This will generate the python file "parsimony_pb2.py".

You can import this file into your favorite python IDE and use it to access the MAT like so:

```
import parsimony_pb2
pb_file = open('input.pb', 'rb')
my_mat = parsimony_pb2.data()
my_mat.ParseFromString(pb_file.read())
pb_file.close()
```

The my_mat object now contains the protobuf information, with general protobuf class attributes and four MAT specific attributes. These are newick, condensed_nodes, metadata, and node_mutations.

The newick attribute is simply the newick string representing the tree, as stored in the protobuf.

```
print(my_mat.newick.count(":"))
print(my_mat.newick[:100])
```

The metadata attribute is a list of metadata message objects, which each have a single attribute which is a list of strings. These strings are the clade annotations for any given node. The mutation list is similar, being a list of lists. Each list contains a series of mutation messages, which have attributes describing their position and identity. Each list corresponds to a single node on the tree.

```
print(len(my_mat.metadata))
print(len(my_mat.node_mutations))
```

Individual node mutations are encoded as integers instead of characters for efficiency. These are in ACGT order- that is, 0 is A, 1 is C, 2 is G, and 3 is T. Additionally, the mut_nuc (new mutation) is another list-like attribute.

```
convert = {i:s for i,s in enumerate("ACGT")}
for t in my_mat.node_mutations:
    if len(t.mutation) > 0:
        first_mut = t.mutation[0]
        print("First mutation encountered identifier string is: {}".format(
            convert[first_mut.ref_nuc] +
            str(first_mut.position) +
            convert[first_mut.mut_nuc[0]]))
        break
```

Condensed nodes is a special format container that essentially acts as a list of objects. Each object has a node_name attribute which is the string naming that node and another container which is essentially a list of strings of sample names.

```
print(my_mat.condensed_nodes[0].node_name)
print(len(my_mat.condensed_nodes[0].condensed_leaves))
```

These are the essentials for writing a custom analysis directly interacting with the protobuf. For most user's purposes, however, matUtils should provide the tools necessary for interacting with a MAT .pb file.

CHAPTER

TEN

PRESENTATIONS

- Covid-19 Dynamics & Evolution 2020
- Stability of SARS-CoV-2 Phylogenetics
- ISMB 2021

CHAPTER

ELEVEN

USHER

UShER is a program for rapid, accurate placement of samples to existing phylogenies. Information on installation, usage, and features can be found *here*. Our manuscript about UShER can be found here.

CHAPTER

TWELVE

MATUTILS

matUtils is a toolkit for querying, interpreting and manipulating the mutation-annotated trees (MATs). Information on its usage can be found *here*.

CHAPTER

THIRTEEN

MATOPTIMIZE

matOptimize is a program to rapidly and effectively optimize a mutation-annotated tree (MAT) for parsimony using subtree pruning and regrafting (SPR) moves within a user-defined radius. Information on its usage can be found *here*.

CHAPTER

FOURTEEN

RIPPLES

RIPPLES is a program that uses a phylogenomic technique to rapidly and sensitively detect recombinant nodes and their ancestors in a mutation-annotated tree (MAT). Information on its usage can be found *here*.

CHAPTER

FIFTEEN

BTE

BTE is a separately packaged Cython API that wraps the highly optimized library underlying these other tools, exposing them for use in Python. Information about its usage can be found *here* and at its repository.