# *treeWAS* Vignette

***Caitlin Collins***

***2017-02-07***

## Introduction

The *treeWAS* R package allows users to apply our phylogenetic tree-based appraoch to Genome-Wide Association Studies (GWAS) to microbial genetic and phenotypic data. *treeWAS* aims to identify genetic variables that are statistically associated with a phenotypic trait of interest and can be applied to data from both bacteria and viruses.

---

## The *treeWAS* Approach

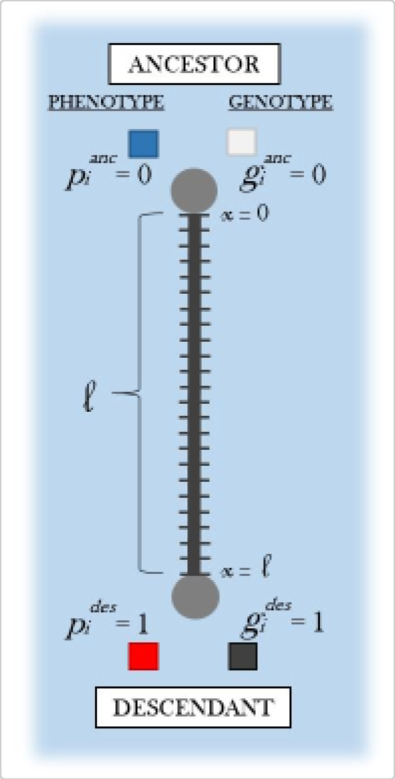
The approach adopted within *treeWAS* uses data simulation to identify a null distribution of association score statistics and disentangle genuine associations, with statistical significance and evolutionary support, from a noisy background of chance associations.

A central aim of *treeWAS* is to control for the confounding effects of population stratification; that is, the overlap between the clonal population structure and the phenotypic distribution which gives rise to spurious associations. This is accomplished through the identification of a null distribution of association score statistics derived from simulated data. Data is simulated in such a way as to maintain both the population stratification and genetic composition of the dataset under analysis, but without recreating the “true” associations beyond those expected to arise from these cofounding factors. The null dataset is simulated using the phylogenetic tree of the real dataset, as well as the original homoplasy distribution including the number of substitutions per site due to both mutation and recombination.

Association score statistics are calculated for both the real and the simulated datasets. The null distribution is drawn from the association scores of the simulated dataset. At the upper tail of this null distribution, a threshold of significance is identified, according to a base p-value corrected for multiple testing. Any loci in the real dataset that have association score values lying above this threshold are deemed to be significantly associated to the phenotype.

### Tests of Association

By default, three measures of association are used to identify significant loci within *treeWAS*.



*Equations below use the following notation:*

- $g$ = Genotypic state…
- $p$ = Phenotypic state…
- $anc$ = … at ancestral nodes
- $des$ = … at descendant nodes
- $n_{term}$ = Number of terminal nodes
- $n_{branch}$ = Number of branches

### Terminal

The `terminal` test solves the following equation, for each genetic locus, at the terminal nodes of the tree only:

$$Terminal = \mid \sum_{i=1}^{n_{term}} \frac{1}{n_{term}} (p_i^{des} g_i^{des} \; - \; (1 - p_i^{des}) g_i^{des} \; - \; p_i^{des}(1 - g_i^{des}) \; + \; (1 - p_i^{des})(1 - g_i^{des})) \mid$$

The `terminal` test is a sample-wide test of association that seeks to identify broad patterns of correlation between genetic loci and the phenotype, without relying on inferences drawn from reconstructions of the ancestral states.

### Simultaneous

The `simultaneous` test solves the following equation, for each genetic locus, across each branch in the tree.

$$Simultaneous = \mid \sum_{i=1}^{n_{branch}} (p_i^{anc} - p_i^{des})(g_i^{anc} - g_i^{des}) \mid$$

This allows for the identification of simultaneous substitutions in both the genetic locus and phenotypic variable on the same branch of the phylogenetic tree (or parallel change in non-binary data). Simultaneous substitutions are an indicator of a deterministic relationship between genotype and phenotype. Moreover, because this score is not negatively impacted by the lack of association on other branches, it may be able to detect associations occurring through complementary pathways (i.e., in some clades but not others).

### Subsequent

The `subsequent` test solves the following equation, for each genetic locus, across each branch in the tree:

$$Subsequent = \left| \sum_{i=1}^{n_{branch}} \left( \frac{4}{3}(p_i^{anc} g_i^{anc}) + \frac{2}{3}(p_i^{anc} g_i^{des}) + \frac{2}{3}(p_i^{des} g_i^{anc}) + \frac{4}{3}(p_i^{des} g_i^{des}) - p_i^{anc} - p_i^{des} - g_i^{anc} - g_i^{des} + 1 \right) \right|$$

Calculating this metric across all branches of the tree allows us to measure in what proportion of tree branches we expect the genotype and phenotype to be in the same state. By drawing on inferences from the ancestral state reconstructions as well as the terminal states, this score may allows us to identify broad, if imperfect, patterns of association.

## Using the *treeWAS* R Package

Here we will go over the functions and arguments used within the *treeWAS* package. To give an example of the code involved, we will use a simple example dataset availabe within *treeWAS*. Integration with the ClonalFrameML software will be discussed in a subsequent section.

## Installation

At present, *treeWAS* is being hosted on GitHub at https://github.com/caitiecollins/treeWAS, though we anticipate that the package will be released on CRAN in the near future.

The most up-to-date version of *treeWAS* can be easily installed directly within R, using the `devtools` package.

```
## install devtools, if necessary:
install.packages("devtools", dep=TRUE)
library(devtools)

## install treeWAS from github:
install_github("caitiecollins/treeWAS/pkg")
library(treeWAS)
```

## Data

To carry out a GWAS using *treeWAS*, the following data is **required**:

### A genetic dataset

A matrix containing binary genetic data (whether this encodes SNPs, gene presence/absence, etc. is up to you). Individuals should be in the rows, and genetic variables in the columns. Both rows and columns must be appropriately labelled.

**A phenotypic variable**

A vector containing either a binary or continuous variable encoding the phenotype for each individual. Each element should have a name that corresponds to the row labels of the genetic data matrix (though the order does not matter).

The following **optional** elements can also be provided by the user or, alternatively, these can be generated automatically by the `treeWAS` function:

**A phylogenetic tree**

An object of class `phylo` (from the *ape* package), containing a phylogenetic tree. Tip labels are required and must correspond to both the row labels of the genetic data matrix and the names of the phenotypic variable (again, the order does not matter).

**An ancestral state reconstruction of the genotype**

A reconstruction of the ancestral states of all loci in the genetic dataset, at all internal nodes. This should include the original states at the terminal nodes and have the same number of columns as the input genetic data matrix. The number of rows should be equal to the total number of nodes in the tree, including the terminal nodes (in rows 1:N). It must be reconstructed with either parsimony or ML (see `?treeWAS`) for consistency with th reconstruction of the ancestral states of the simulated genetic dataset within *treeWAS*.

**An ancestral state reconstruction of the phenotype**

A reconstruction of the ancestral states of the phenotypic variable at all nodes in the tree, including the original states at the terminal nodes (at positions 1:N).

We will use the data stored within *treeWAS* as an example throughout this section of the vignette. We load the data using the `data` function and examine its structure below:

```
## Load example data:
data(snps)
data(phen)
data(tree)

## Examine data:
## genetic data
str(snps)
```

```
##  num [1:100, 1:20003] 0 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:100] "A_1090" "B_1078" "A_1070" "B_1083" ...
##   ..$ : chr [1:20003] "1.g" "1.a" "2.t" "2.g" ...
## phenotype
str(phen)
```

```
##  Factor w/ 2 levels "A","B": 2 2 2 2 1 2 1 1 2 2 ...
##  - attr(*, "names")= chr [1:100] "B_1132" "B_1105" "B_1079" "B_1133" ...
```

```
table(phen)
```

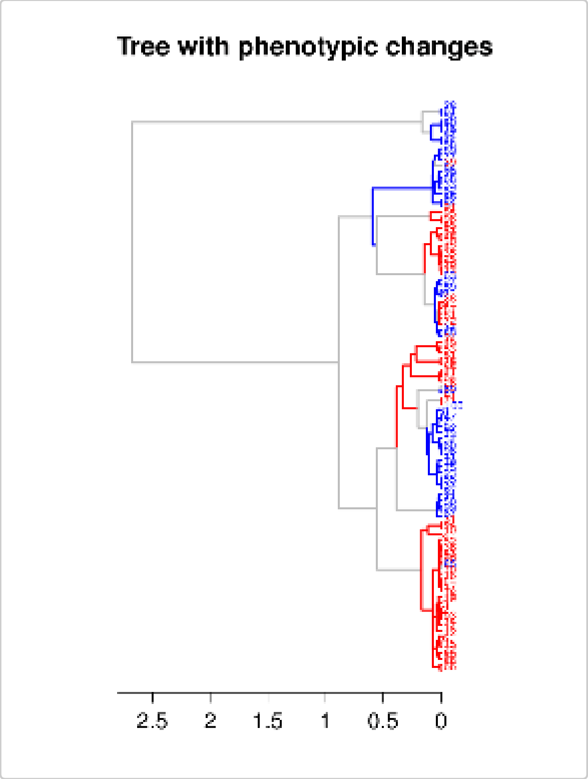
```
## phen
##  A  B
## 45 55
```

```
## tree
str(tree)
```

```
## List of 4
##  $ edge       : int [1:198, 1:2] 167 167 150 150 168 168 184 184 190 190 ...
##  $ tip.label  : chr [1:100] "B_1132" "B_1105" "B_1079" "B_1133" ...
##  $ edge.length: num [1:198] 0.00884 0.00884 0.01584 0.01584 0.00863 ...
##  $ Nnode      : int 99
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "pruningwise"
```

```
## Load colours:
data(phen.plot.col)

## Plot tree showing phenotype:
plot.phen(tree, phen.nodes=phen.plot.col$all.nodes)
```



# Data Cleaning

Before entering the genetic and phenotypic data into the `treeWAS` function, the user must ensure that the data is in the appropriate format.

## Labels:

The genetic data matrix, phenotype, and phylogenetic tree (terminal nodes) should all be labelled with corresponding sets of names for the individuals.

Any individual that is not present in one or more of the genetic data matrix, the phenotypic variable, and/or the phylogenetic tree must be removed.

```
## Check that labels are present:
is.null(tree$tip.label)
```

```
## [1] FALSE
```

```
is.null(rownames(snps))
```

```
## [1] FALSE
```

```
is.null(names(phen))
```

```
## [1] FALSE
```

```
## Cross-check labels with each other:
all(tree$tip.label %in% rownames(snps))
```

```
## [1] TRUE
```

```
all(rownames(snps) %in% tree$tip.label)
```

```
## [1] TRUE
```

```
all(tree$tip.label %in% names(phen))
```

```
## [1] TRUE
```

```
all(names(phen) %in% tree$tip.label)
```

```
## [1] TRUE
```

```
all(names(phen) %in% rownames(snps))
```

```
## [1] TRUE
```

```
all(rownames(snps) %in% names(phen))
```

```
## [1] TRUE
```

The above checks are run within `treeWAS`, but it may be worthwhile to confirm for yourself that all labels are present and look correct.

### Biallelic loci:

If, in the genetic data matrix, redundant columns are present for binary loci (ie. Denoting the state of the second allele as the inverse of the previous colummn), these should be removed. For loci with more than two alleles, all columns should be retained.

The removal of redundant binary columns can be done by hand; but, the function `get.binary.snps` may also be used. This function requires column names to have a two-character suffix and unique locus identifiers. The function expects the suffixes ".a", ".c", ".g", ".t" (e.g., "Locus_123243.a", "Locus_123243.g"), though alternative two-character suffixes can be used (e.g., "Locus_123243_1", "Locus_123243_2") by setting the argument `force = TRUE`.

Please also be careful not to accidentally remove any purposeful duplications with repeated names; for example, if you have deliberately duplicated columns without subsequently generating unique column names (e.g., by expanding unique columns according to an index returned by ClonalFrameML).

Before running the `get.binry.snps` function, you can check the suffixes of the genetic data matrix's column names using another small function in *treeWAS*, `keepLastN`, which allows you to keep the last N characters of a variable or vector:

```
suffixes <- keepLastN(colnames(snps), n = 2)
suffixes <- unique(suffixes)

all(suffixes %in% c(".a", ".c", ".g", ".t"))
```

```
## [1] TRUE
```

```
snps <- get.binary.snps(snps)
```

### Missing data:

Missing data is permitted (denoted by `NA` values only) in the genetic data matrix, but if more than 50% of any column is composed of NAs, we recommend that this column be removed from the dataset. This will be done automatically within `treeWAS`. If, for some reason, you do not wish this to be the case, set the argument `na.rm` to `FALSE`.

## Arguments

The `treeWAS` function takes the following arguments:

```
## Don't run this:
out <- treeWAS(snps,
               phen,
               tree = c("UPGMA", "nj", "ml"),
               n.subs = NULL,
               sim.n.snps = ncol(snps)*10,
               test = c("terminal", "simultaneous", "subsequent"),
               snps.reconstruction = "parsimony",
               snps.sim.reconstruction = "parsimony",
               phen.reconstruction = "parsimony",
               na.rm = TRUE,
               p.value = 0.01,
               p.value.correct = c("bonf", "fdr", FALSE),
               p.value.by = c("count", "density"),
               dist.dna.model = "JC69",
               plot.tree = FALSE,
               plot.manhattan = TRUE,
               plot.null.dist = TRUE,
               plot.dist = FALSE,
               snps.assoc = NULL,
               filename.plot = NULL,
               seed = NULL)
```

`snps` : A matrix containing binary genetic data, with individuals in the rows and genetic loci in the columns and both rows and columns labelled.

`phen` : A vector containing the phenotypic state of each individual, whose length is equal to the number of rows in `snps` and which is named with the same set of labels. The phenotype can be either binary (character or numeric) or continuous (numeric).

`tree` : A `phylo` object containing the phylogenetic tree; or, a character string, one of `"nj"`, `"ml"`, or `"UPGMA"` (the default), specifying the method of phylogenetic reconstruction.

`n.subs` : A numeric vector containing the homoplasy distribution (if known, see details), or NULL (the default).

`sim.n.snps` An integer specifying the number of loci to be simulated for estimating the null distribution (by default `10*ncol(snps)`).

`test` A character string or vector containing one or more of the following available tests of association: `"terminal"`, `"simultaneous"`, `"subsequent"`, `"cor"`, `"fisher"`. By default, the first three tests are run (see details).

`snps.reconstruction` Either a character string specifying `"parsimony"` (the default) or `"ML"` (maximum likelihood) for the ancestral state reconstruction of the genetic dataset, or a matrix containing this reconstruction if it has been performed elsewhere.

`snps.sim.reconstruction` A character string specifying `"parsimony"` (the default) or `"ML"` (maximum likelihood) for the ancestral state reconstruction of the simulated null genetic dataset.

`phen.reconstruction` Either a character string specifying `"parsimony"` (the default) or `"ML"` (maximum likelihood) for the ancestral state reconstruction of the phenotypic variable, or a vector containing this reconstruction if it has been performed elsewhere.

`p.value` A number specifying the base p-value to be set the threshold of significance (by default, `0.01`).

`p.value.correct` A character string, either `"bonf"` (the default) or `"fdr"`, specifying whether correction for multiple testing should be performed by Bonferonni correction (recommended) or the False Discovery Rate.

`p.value.by` A character string specifying how the upper tail of the p-value distribution is to be identified. Either `"count"` (the default, recommended) for a simple count-based approach or `"density"` for a kernel-density based approximation.

`dist.dna.model` A character string specifying the type of model to use in reconstructing the phylogenetic tree for calculating the genetic distance between individual genomes, only used if `tree` is a character string (see ?dist.dna).

`plot.tree` A logical indicating whether to generate a plot of the phylogenetic tree (`TRUE`) or not (`FALSE`, the default).

`plot.manhattan` A logical indicating whether to generate a manhattan plot for each association score (`TRUE`, the default) or not (`FALSE`).

`plot.null.dist` A logical indicating whether to plot the null distribution of association score statistics (`TRUE`, the default) or not (`FALSE`).

`plot.dist` A logical indicating whether to plot the true distribution of association score statistics (`TRUE`) or not (`FALSE`, the default).

`snps.assoc` An optional character string or vector specifying known associated loci to be demarked in results plots (e.g., from previous studies or if data is simulated); else NULL.

`filename.plot` An optional character string denoting the file location for saving any plots produced; else `NULL`.

`seed` An optional integer to control the pseudo-randomisation process and allow for identical repeat runs of the function; else `NULL`.

## Running treeWAS

Running `treeWAS` takes only one function. It should finish within a couple of minutes, depending on the size of the dataset.

```
out <- treeWAS(snps = snps,
               phen = phen,
               tree = tree,
               n.subs = NULL,
               sim.n.snps = ncol(snps)*10,
               test = c("terminal", "simultaneous", "subsequent"),
               snps.reconstruction = "parsimony",
               snps.sim.reconstruction = "parsimony",
               phen.reconstruction = "parsimony",
               na.rm = TRUE,
               p.value = 0.01,
               p.value.correct = "bonf",
               p.value.by = "count",
               dist.dna.model = "JC69",
               plot.tree = FALSE,
               plot.manhattan = TRUE,
               plot.null.dist = TRUE,
               plot.dist = FALSE,
               snps.assoc = NULL,
               filename.plot = NULL,
               seed = 1)
```

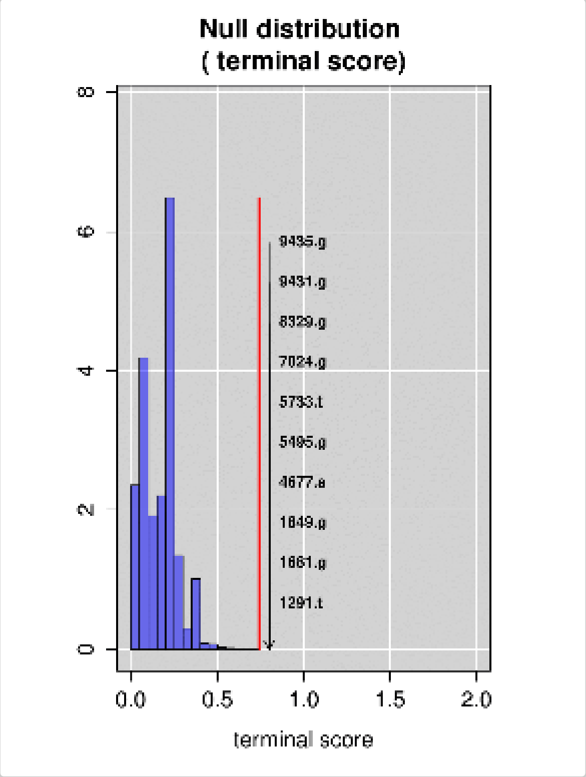
For large datasets, if you find you are running into errors relating to `memory.size`, try these suggestions:

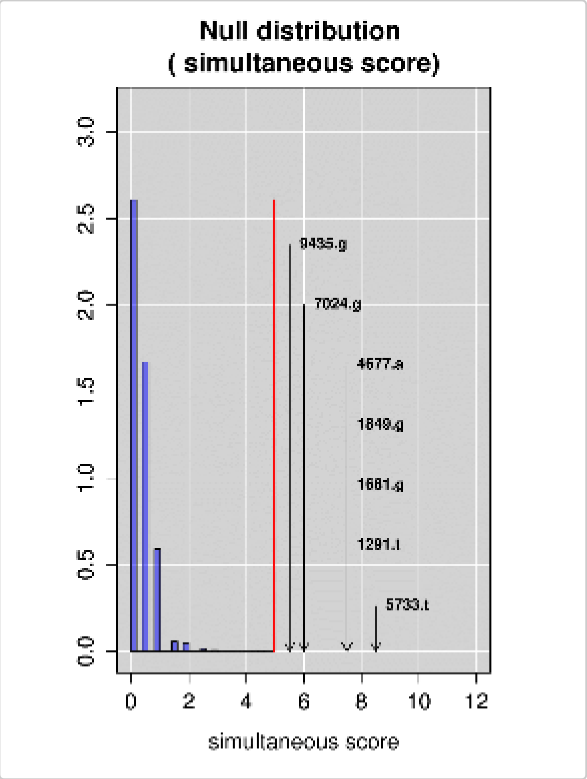
1. Run `treeWAS` on a computer with more memory, if you have access to one.
2. Restart your computer, open R and run `treeWAS` before opening any other programs.
3. If necessary, gradually reduce the number of simulated loci (`sim.n.snps`), e.g., to `5*ncol(snps)`.

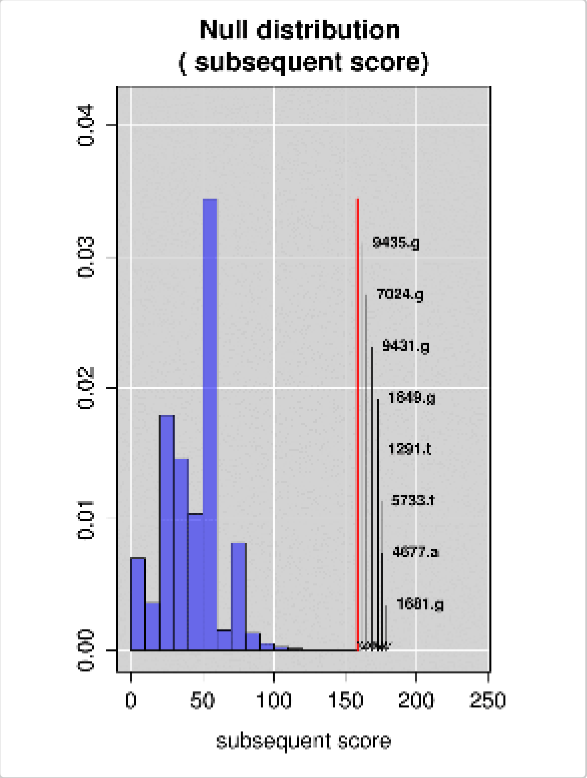
## Interpreting Output

If `plot.null.dist` is set to `TRUE`, the null distributions and findings will be plotted for each association score. If `plot.dist` is `TRUE`, a distribution of the values will be plotted for each association score. And if `plot.manhattan` is set to `TRUE`, a manhattan plot will show the values for each each association score, with a threshold delineating the significant findings from the insignificant.

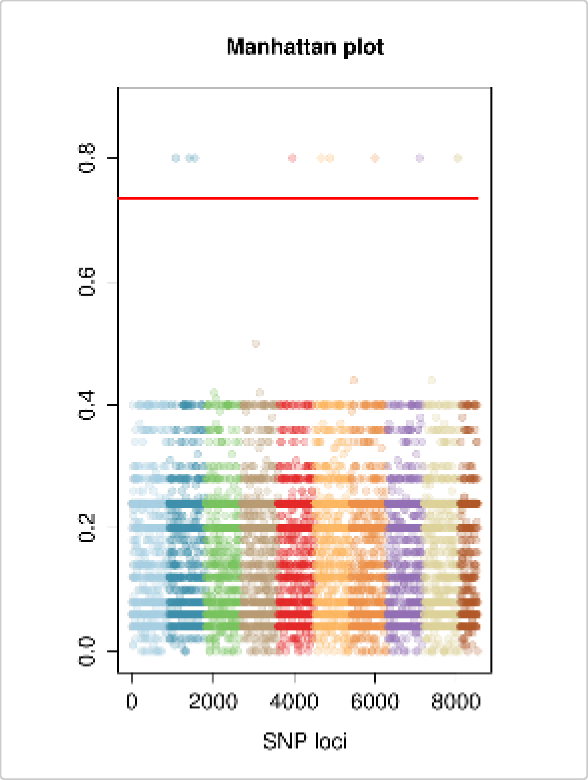
By default, `plot.null.dist` is set to `TRUE`.

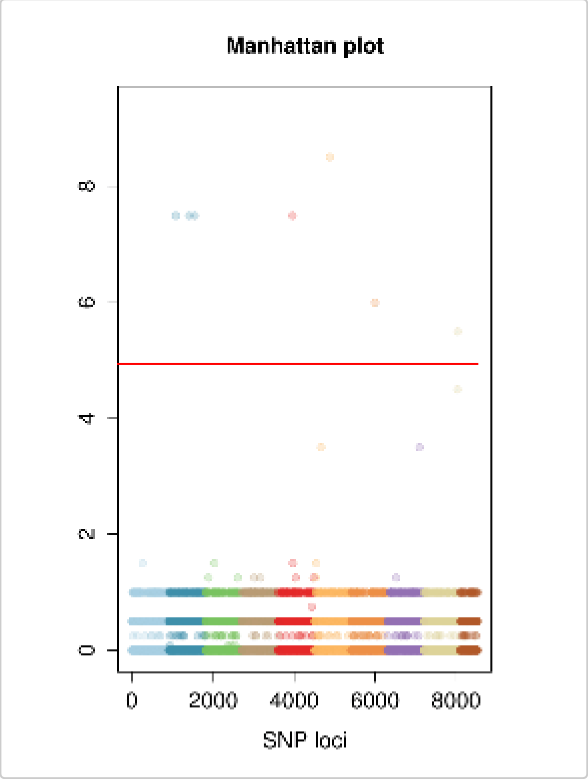


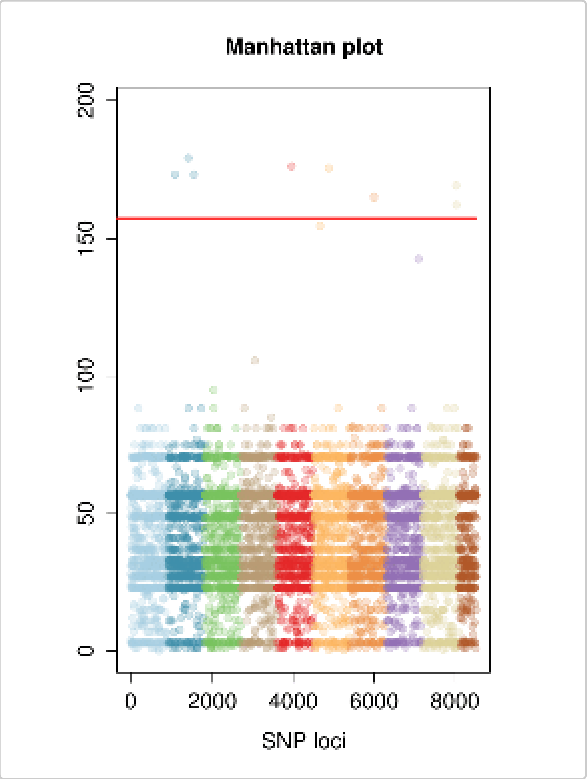




By default, `plot.manhattan` is also set to `TRUE`.


Manhattan plot
0.0
0.2
0.4
0.6
0.8
0
2000
4000
6000
8000
SNP loci


Manhattan plot
0
2
4
6
8
0
2000
4000
6000
8000
SNP loci



The `treeWAS` function returns a list containing one element comprising the complete pooled set of findings, as well as one element for each of the association scores applied (by default, three).

Each list element contains the following:

`corr.dat`
The association score values associated with the loci in the real genetic dataset. `corr.sim`

The association score values associated with the loci in the simulated null dataset. `p.vals`

The p-values associated with the loci in the real genetic dataset for this association score. `sig.thresh`

The threshold of significance. `sig.snps`

A data frame describing the genetic loci identified as significant. `min.p.value`

The minimum p-value. P-values listed as zero can only truly be defined as below this value.

Because `treeWAS` returns a large volume of data, it is recommended that the `print.treeWAS` function is used to examine the set of results identified:

```
print(out, sort.by.p=FALSE)
```

```
## ####################
## ## treeWAS output ##
## ####################
##
## ####################
## ## All findings:  ##
## ####################
## Number of significant loci: [1] 10
## Significant loci:
##  [1] "1291.t" "1681.g" "1849.g" "4677.a" "5495.g" "5733.t" "7024.g"
##  [8] "8329.g" "9431.g" "9435.g"
##
## ########################
## ## Findings by test:  ##
##  ########################
##   ####################
##   ##  terminal test ##
##   ####################
## Number of significant loci:
## [1] 10
## Significance threshold:
## 99.99996%
##      0.74
## Significant loci:
##        SNP.locus p.value Test.statistic S1P1 S0P0 S1P0 S0P1
## 1291.t      1072       0            0.8   49   41    4    6
## 1681.g      1399       0            0.8   49   41    4    6
## 1849.g      1539       0            0.8   49   41    4    6
## 4677.a      3958       0            0.8    3    7   38   52
## 5495.g      4661       0            0.8   48   42    3    7
## 5733.t      4875       0            0.8   50   40    5    5
## 7024.g      5994       0            0.8   49   41    4    6
## 8329.g      7094       0            0.8   48   42    3    7
## 9431.g      8028       0            0.8   50   40    5    5
## 9435.g      8032       0            0.8   51   39    6    4
##   ########################
##   ##  simultaneous test ##
##   ########################
## Number of significant loci:
## [1] 7
## Significance threshold:
## 99.99996%
##  4.980434
## Significant loci:
##        SNP.locus p.value Test.statistic S1P1 S0P0 S1P0 S0P1
## 1291.t      1072       0            7.5   49   41    4    6
## 1681.g      1399       0            7.5   49   41    4    6
## 1849.g      1539       0            7.5   49   41    4    6
## 4677.a      3958       0            7.5    3    7   38   52
## 5733.t      4875       0            8.5   50   40    5    5
## 7024.g      5994       0            6.0   49   41    4    6
## 9435.g      8032       0            5.5   51   39    6    4
##   ######################
##   ##  subsequent test ##
##   ######################
## Number of significant loci:
## [1] 8
## Significance threshold:
## 99.99996%
##  158.3797
## Significant loci:
##        SNP.locus p.value Test.statistic S1P1 S0P0 S1P0 S0P1
## 1291.t      1072       0       173.0000   49   41    4    6
## 1681.g      1399       0       179.0000   49   41    4    6
## 1849.g      1539       0       173.0000   49   41    4    6
## 4677.a      3958       0       176.0000    3    7   38   52
## 5733.t      4875       0       175.3333   50   40    5    5
## 7024.g      5994       0       165.0000   49   41    4    6
## 9431.g      8028       0       169.0000   50   40    5    5
## 9435.g      8032       0       162.3333   51   39    6    4
```

## Integration with ClonalFrameML

The *treeWAS* R package has also been designed to work with the output of ClonalFrameML. By using the simple `read.CFML` function, you can convert the output of ClonalFrameML into a form suitable for input into `treeWAS`.

For practice using ClonalFrameML, download the ClonalFrameML example data and follow the steps on the wiki to arrive at the `example.output` dataset. To run the example below, replace the prefix with the path to the `example.output` dataset on your computer or to another set of output from ClonalFrameML containing:

1. “prefix.labelled_tree.newick”
2. “prefix.ML_sequence.fasta”
3. “prefix.position_cross_reference.txt”

To convert the data, use the `read.CFML` function:

```
prefix <- "./exmple.output"
dat <- read.CFML(prefix=prefix)
```

Isolate the elements of the output of `read.CFML`:

```
tree <- foo$tree
seqs <- foo$seqs
index <- foo$index
dist <- foo$dist
```

Convert the `DNAbin` object containing the genetic dataset into a binary `snps` matrix, and expand the unique columns back into the entire dataset:

```
## Convert DNAbin object:
snps.rec <- DNAbin2genind(seqs, polyThres=0)
snps.rec <- snps.rec@tab

## Get binary loci:
snps.rec.bin <- get.binary.snps(snps.rec)
str(snps.rec.bin)

## Expand snps with index:
snps.rec <- snps.bin
snps.rec <- snps.rec[,index]
```

Note that the sequences returned by ClonalFrameML contain the reconstructed states at internal nodes. To get the loci for only the terminal nodes, as is required to be input into `treeWAS`, just keep the first N rows:

```
N <- nrow(snps.rec)-tree$Nnode
toKeep <- 1:N
snps <- snps.rec[toKeep, ]
```

You can now run `treeWAS` using this `snps` matrix, and the `tree` returned from `read.CFML`.

Instead of supplying `snps.rec` to the argument `snps.reconstruction`, it may be worth (re-)creating the ancestral state reconstruction within . This is recommended, provided the dataset you are working with is not very large, in case any inconsistencies exist between the reconstruction performed by ClonalFrameML and that run within `treeWAS` (which will also be applied to the simulated null dataset).

The `dist` object can be provided in the `n.subs` argument, though you may also leave this `NULL` and have `treeWAS` recreate `n.subs` internally, again in case there is any discrepancy between the internal and external methods or reconstruction.

All you would need to run `treeWAS` with this example dataset is a phenotype. We can make a toy phenotype for this purpose, although no significant findings should result:

```
set.seed(1)
phen <- sample(c(0,1), nrow(snps), replace=TRUE)
phen <- as.factor(phen)
names(phen) <- rownames(snps)

## Exmine toy phenotype
str(phen)
table(phen)
```

You can now run `treeWAS` using the converted ClonalFrameML output:

```
## Example not run...
out <- treeWAS(snps = snps,
                phen = phen,
                tree = tree,
                n.subs = NULL,
                sim.n.snps = ncol(snps)*10,
                test = c("terminal", "simultaneous", "subsequent"),
                snps.reconstruction = "parsimony",
                snps.sim.reconstruction = "parsimony",
                phen.reconstruction = "parsimony",
                na.rm = TRUE,
                p.value = 0.01,
                p.value.correct = "bonf",
                p.value.by = "count",
                dist.dna.model = "JC69",
                plot.tree = FALSE,
                plot.manhattan = TRUE,
                plot.null.dist = TRUE,
                plot.dist = FALSE,
                snps.assoc = NULL,
                filename.plot = NULL,
                seed = 1)
```

# Bugs & Features

## Report a Bug

The *treeWAS* R package is still in its early days. As such, you may encounter an error before we do. If you believe you have identified a bug, or you are unable to overcome an error after consulting the documentation, we encourage you to visit our Issues Page. If your issue has not been documented by another user, please Post A New Issue.

## Request a Feature

There may be a feature absent from *treeWAS* that you wish could be performed by the R package. We welcome your suggestions and invite you to Place a Feature Request on our Issues Page.

We are working towards a number of features in the near future, for example, allowing *treeWAS* to handle non-binary categorical phenotypes.