Package ‘admixturegraph’

December 13, 2016

Type Package

Title Admixture Graph Manipulation and Fitting

Version 1.0.2

Date 2016-12-12

Description Implements tools for building and visualising admixture graphs
and for extracting equations from them. These equations can be compared to f-statistics obtained from data to test the consistency of a graph against data --
for example by comparing the sign of f_4-statistics with the signs predicted by
the graph -- and graph parameters (edge lengths and admixture proportions) can
be fitted to observed statistics.

License GPL-2

Depends R (>= 3.2.2)

Imports doParallel, dplyr, foreach, ggplot2, graphics, MASS, methods,
neldermead, parallel, pracma, stats, utils, grDevices

Suggests knitr, rmarkdown, testthat

URL `https://github.com/mailund/admixture_graph`

BugReports `https://github.com/mailund/admixture_graph/issues`

VignetteBuilder knitr

LazyData true

RoxygenNote 5.0.1

NeedsCompilation no

Author Thomas Mailund [cre, aut],
Kalle Leppala [aut],
Svend Nielsen [aut]

Maintainer Thomas Mailund <`mailund@birc.au.dk`>

Repository CRAN

Date/Publication 2016-12-13 15:33:28

R topics documented:

admixturegraph-package . . . 4
add_an_admixture . . . 5
add_an_admixture2 . . . 7
add_a_leaf . . . 8
add_graph_f4 . . . 9
add_graph_f4_sign . . . 10
admixture_edge . . . 10
admixture_proportions . . . 11
admix_props . . . 11
agraph . . . 12
agraph_children . . . 13
agraph_parents . . . 13
agraph_weights . . . 14
all_graphs . . . 14
all_paths . . . 15
all_path_overlaps . . . 15
bears . . . 16
build_edge_optimisation_matrix . . . 16
burn_in . . . 17
calculate_concentration . . . 17
canonise_expression . . . 18
canonise_graph . . . 18
coef.agraph_fit . . . 19
cost_function . . . 19
edge . . . 20
edge_optimisation_function . . . 21
eight_leaves_trees . . . 22
evaluate_f4 . . . 23
examine_edge_optimisation_matrix . . . 23
extract_admixture_proportion_parameters . . . 24
extract_graph_parameters . . . 25
extract_trees . . . 25
f2 . . . 26
f3 . . . 26
f4 . . . 27
f4stats . . . 27
fast_fit . . . 28
fast_plot . . . 30
filter_on_leaves . . . 31
fitted.agraph_fit . . . 31
fit_graph . . . 32
fit_graph_list . . . 34
fit_permutations_and_graphs . . . 35
five_leaves_graphs . . . 36
format_path . . . 37
four_leaves_graphs . . . 38

get_graph_f4_sign . . . 39
graphs_2_0 . . . 39
graphs_3_0 . . . 40
graphs_3_1 . . . 40
graphs_4_0 . . . 41
graphs_4_1 . . . 41
graphs_4_2 . . . 42
graphs_5_0 . . . 42
graphs_5_1 . . . 43
graphs_5_2 . . . 43
graphs_6_0 . . . 44
graphs_6_1 . . . 44
graphs_6_2 . . . 45
graphs_7_0 . . . 45
graphs_7_1 . . . 46
graphs_8_0 . . . 46
graph_environment . . . 47
graph_to_vector . . . 47
is_descendant_of . . . 48
is_negative . . . 48
is_positive . . . 49
is_unknown . . . 49
is_zero . . . 49
log_likelihood . . . 50
log_sum_of_logs . . . 51
make_an_outgroup . . . 51
make_mcmc_model . . . 52
make_permutations . . . 53
model_bayes_factor_n . . . 54
model_likelihood . . . 55
model_likelihood_n . . . 55
mynonneg . . . 56
no_admixture_events . . . 57
no_admixture_events.agraph . . . 57
no_admixture_events.agraph_fit . . . 58
no_admixture_events.agraph_fit_list . . . 58
no_poor_fits . . . 59
no_poor_fits.agraph_fit . . . 59
no_poor_fits.agraph_fit_list . . . 60
overlaps_sign . . . 60
parent_edges . . . 61
path_overlap . . . 61
plot.agraph . . . 62
plot.agraph_fit . . . 64
plot.f4stats . . . 65
plot_fit_1 . . . 65
plot_fit_2 . . . 66
poor_fits . . . 67

poor_fits.agraph_fit . . . 67
poor_fits.agraph_fit_list . . . 68
print.agraph_fit . . . 68
project_to_population . . . 69
remove_duplicates . . . 69
rename_nodes . . . 70
residuals.agraph_fit . . . 70
run_metropolis_hasting . . . 71
seven_leaves_graphs . . . 72
seven_leaves_trees . . . 73
sf2 . . . 73
sf3 . . . 74
sf4 . . . 74
six_leaves_graphs . . . 75
split_population . . . 76
split_population.agraph_fit . . . 76
split_population.data.frame . . . 77
summary.agraph_fit . . . 77
sum_of_squared_errors . . . 78
sum_of_squared_errors.agraph_fit . . . 78
sum_of_squared_errors.agraph_fit_list . . . 79
thinning . . . 79
vector_to_graph . . . 80

Index **81**

`admixturegraph-package`

admixturegraph: Visualising and analysing admixture graphs.

Description

The package provides functionality to analyse and test admixture graphs against the f statistics described in the paper Ancient Admixture in Human History, Patterson *et al.*, Genetics, Vol. 192, 1065–1093, 2012.

Details

The f statistics – f_2, f_3, and f_4 – extract information about correlations between gene frequencies in different populations (or single diploid genome samples), which can be informative about patterns of gene flow between these populations in form of admixture events. If a graph is constructed as a hypothesis for the relationship between the populations, equations for the expected values of the f statistics can be extracted, as functions of edge lengths – representing genetic drift – and admixture proportions.

This package provides functions for extracting these equations and for fitting them against computed f statistics. It does not currently provide functions for computing the f statistics – for that we refer to the ADMIXTOOLS software package.

`add_an_admixture` *Adds a new admixture event to a graph.*

Description

Given an admixture graph, selects a child edge and a parent edges and adds a new edge from the parent edge to the child edge with an admixture event, if possible. Thus, the resulting graph is an extension of the input graph in the sense that erasing one of the admixture edges (the new one) we get the original admixture graph. However, we know that in practice when fitting data to admixture graphs, the best graph with k admixture events is not always an extension like that from the best graph with $k-1$ admixture events. For a more relaxed way of adding a new admixture event, try `add_an_admixture2`.

Usage

```
add_an_admixture(graph, admixture_variable_name, labels_matter = FALSE,
  outgroup = "")
```

Arguments

`graph`	An admixture graph.
`admixture_variable_name`	A name for the new admixture proportion.
`labels_matter`	When `FALSE` (the default value), we consider two admixture graphs similar when they have the same topology but permuted admixture proportion names. When `TRUE`, the already existing admixture events and the edges leading to them are considered labeled.
`outgroup`	An optional parameter for the preferred outgroup.

Value

A list of graphs made by adding a new admixture event to the input graph. The list has no duplicate elements (what that means depends on the value of `labels_matter`).

See Also

`all_graphs`

`fit_permutations_and_graphs`

`fit_graph_list`

`add_a_leaf`

`add_an_admixture2`

`make_an_outgroup`

Examples

```
# To illustrate what the parameter labels_matter does, consider the following graph:

leaves <- c("A", "B", "C")
inner_nodes <- c("R", "x", "y", "M")
edges <- parent_edges(c(edge("x", "R"),
                        edge("y", "R"),
                        edge("A", "x"),
                        edge("B", "M"),
                        edge("C", "y"),
                        admixture_edge("M", "x", "y")))
admixtures <- admixture_proportions(c(admix_props("M", "x", "y", "p")))
graph <- agraph(leaves, inner_nodes, edges, admixtures)
plot(graph, show_admixture_labels = TRUE, title = "graph")

# There are 15 ways this graph can be extended to a graph with two admixture events by
# adding an admixture edge, as can be seeing by having the program explicitly construct
# all the cases:

short_list <- add_an_admixture(graph, "q")
print(length(short_list))

# However, maybe we already have a specific historical event in mind corresponding to the
# original admixture event in graph, or a fixed value for the admixture proportion p.
# Then, for example, it makes a difference to us whether we consider the possibility of
# another admixture event occurring before that event,

leaves <- c("A", "B", "C")
inner_nodes <- c("R", "x", "y", "z", "M", "N")
edges <- parent_edges(c(edge("x", "R"),
                        edge("z", "R"),
                        edge("y", "z"),
                        edge("A", "x"),
                        edge("B", "M"),
                        edge("C", "y"),
                        admixture_edge("N", "x", "z"),
                        admixture_edge("M", "N", "y")))
admixtures <- admixture_proportions(c(admix_props("N", "x", "z", "q"),
                                      admix_props("M", "N", "y", "p")))
example1 <- agraph(leaves, inner_nodes, edges, admixtures)
plot(example1, show_admixture_labels = TRUE, title = "example 1")

# or after that event,

leaves <- c("A", "B", "C")
inner_nodes <- c("R", "x", "y", "z", "M", "N")
edges <- parent_edges(c(edge("x", "R"),
                        edge("y", "R"),
                        edge("z", "y"),
                        edge("A", "x"),
                        edge("B", "N"),
                        edge("C", "z"),
                        admixture_edge("M", "x", "y"),
                        admixture_edge("N", "M", "z")))
admixtures <- admixture_proportions(c(admix_props("M", "x", "y", "p"),
                                      admix_props("N", "M", "z", "q")))
example2 <- agraph(leaves, inner_nodes, edges, admixtures)
plot(example2, show_admixture_labels = TRUE, title = "example 2")

# even though as (acyclic) directed graphs with labeled leaves example 1
# and example 2 are isomorphic.
# Counting cases like that dirrerent, we get 21 possible extensions:

long_list <- add_an_admixture(graph, "q", labels_matter = TRUE)
print(length(long_list))
```

`add_an_admixture2` *Adds a new admixture event to a graph.*

Description

Given an admixture graph, selects a child edge and two parent edges, disconnects the child edge from its original parent node and connects it to the two parent edges with an admixture event, if possible. Thus, contrary to `add_an_admixture`, the resulting graph need not be an extension of the input graph in the sense that erasing one of the admixture edges we get the original admixture graph. In practice, we know that when fitting data to admixture graphs, the best graph with k admixture events is not always an extension like that from the best graph with $k - 1$ admixture events. Most likely it doesn't need to be an extension like this (the two new admixture edges can both go where ever) either.

Usage

```
add_an_admixture2(graph, admixture_variable_name, outgroup = "")
```

Arguments

`graph`	An admixture graph.
`admixture_variable_name`	A name for the new admixture proportion.
`outgroup`	An optional parameter for the preferred outgroup.

Value

A list of graphs made by adding a new admixture event to the input graph. The list contains duplicate elements, and may even contain graphs with *eyes* (two inner nodes with the property that all the paths between any two leaves visits both or neither of them).

See Also

all_graphs

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

make_an_outgroup

---

add_a_leaf *Adds a new leaf to a graph.*

Description

Given an admixture graph, selects an edge and branches off a new edge ending at a new leaf.

Usage

```
add_a_leaf(graph, leaf_name, outgroup = "")
```

Arguments

`graph`	An admixture graph.
`leaf_name`	A name for the new leaf.
`outgroup`	An optional parameter for the preferred outgroup, which can be the new leaf.

Value

A list of graphs made by adding a new leaf to the input graph. The list has no duplicate elements.

See Also

all_graphs

fit_permutations_and_graphs

fit_graph_list

add_an_admixture

add_an_admixture2

make_an_outgroup

Examples

```
# Take a look at how much trees there are:
leaves <- c("1", "2")
inner_nodes <- c("R")
edges <- parent_edges(c(edge("1", "R"), edge("2", "R")))
admixtures <- NULL
Lambda <- agraph(leaves, inner_nodes, edges, admixtures)
set <- list(Lambda)
for (i in seq(1, 6)) {
  new_set <- list()
  for (tree in set) {
    new_set <- c(new_set, add_a_leaf(tree, paste(i + 2)))
  }
  set <- new_set
  cat("There are ")
  cat(length(set))
  cat(" different trees with ")
  cat(i + 2)
  cat(" labeled leaves.")
  cat("\n")
}
# In general, there are 1*3*5*...*(2n - 5) different trees with n labeled leaves
# (A001147 in the online encyclopedia of integer sequences).
# Exhaustive search through the graph space is hard!
```

`add_graph_f4` *Evalutes the f_4 statistics for all rows in a data frame and extends the data frame with the graph f_4.*

Description

The data frame, `data`, must contain columns `W`, `X`, `Y`, and `Z`. The function then computes the $f_4(W, X; Y, Z)$ statistics (also known as the D statistics) for all rows and adds these as a column, `graph_f4`, to the data frame.

Usage

```
add_graph_f4(data, graph, env)
```

Arguments

`data`	The data frame to get the labels to compute the f_4 statistics from.
`graph`	The admixture graph.
`env`	The environment to evaluate the f_4 statistics in.

Value

A data frame identical to `data` except with an additional column, `graph_f4`, containing the f_4 values as determined by the graph and the environment.

`add_graph_f4_sign` *Extend a data frame with f_4 statistics predicted by a graph.*

Description

Extracts the sign for the f_4 statistics predicted by the graph for all rows in a data frame and extends the data frame with the graph f_4.

Usage

```
add_graph_f4_sign(data, graph)
```

Arguments

`data`	The data frame to get the labels to compute the f_4 statistics from.
`graph`	The admixture graph.

Details

The data frame, `data`, must contain columns `W`, `X`, `Y`, and `Z`. The function then computes the sign of the $f4(W, X; Y, Z)$ statistics for all rows and adds these as a column, `graph_f4_sign`, to the data frame.

Value

A data frame identical to `data` except with an additional column, `graph_f4_sign`, containing the sign of the f_4 statistics as determined by the graph.

`admixture_edge` *Create an admixture edge from a child to two parents.*

Description

Syntactic suggar for constructing edges in an admixture graph.

Usage

```
admixture_edge(child, parent1, parent2, prop = NA)
```

Arguments

`child`	The name of the child node.
`parent1`	The name of the parent node.
`parent2`	The name of the parent node.
`prop`	Admixture proportions from `parent1` to `child`. If this parameter is not provided, you must explicitly specify the admixture proportion parameters in the `agraph` function call.

`admixture_proportions` *Create the list of admixture proportions for an admixture graph.*

Description

Syntactic suggar for constructing edges in an admixture graph.

Usage

```
admixture_proportions(admix_props)
```

Arguments

`admix_props`	The admixture proportions.

`admix_props` *Specify the proportions in an admixture event.*

Description

Syntactic suggar for constructing edges in an admixture graph.

Usage

```
admix_props(child, parent1, parent2, prop)
```

Arguments

`child`	The child node.
`parent1`	The first parent.
`parent2`	The second parent.
`prop`	The admixture proportions coming from the first parent.

`agraph` *Create an admixture graph object.*

Description

Create an admixture graph object, an acyclic directed graph.

Usage

```
agraph(leaves, inner_nodes, parent_edges,
  admixture_proportions = extract_admixture_proportion_parameters(parent_edges))
```

Arguments

`leaves`	The names of the leaves in the admixture graph. Do not use `(, )` or a single `R`.
`inner_nodes`	The name of the inner nodes in the admxture graph. Do not use `(, )` or a single `R` except for the root if you wish.
`parent_edges`	The list of edges in the graph, created by `parent_edges`.
`admixture_proportions`	The list of admixture proportions; created by `admixture_proportions`. Do not use `+, -, *, (, )`.

Value

An admixture graph object.

See Also

`edge`

`admixture_edge`

`admix_props`

`parent_edges`

`admixture_proportions`

`plot.agraph`

Examples

```
leaves <- c("A", "B", "C", "D")
inner_nodes <- c("ab", "b", "bc", "abc", "abcd")
edges <- parent_edges(c(edge("A", "ab"),
                        edge("B", "b"),
                        edge("C", "bc"),
                        edge("D", "abcd"),
                        edge("ab", "abc"),
                        edge("bc", "abc"),
                        edge("abc", "abcd"),
                        admixture_edge("b", "ab", "bc")))
admixtures <- admixture_proportions(c(admix_props("b", "ab", "bc", "x")))

graph <- agraph(leaves, inner_nodes, edges, admixtures)
```

agraph_children *Build the child incidene matrix from an parent edge list.*

Description

Build the child incidene matrix from an parent edge list.

Usage

```
agraph_children(nodes, parent_edges)
```

Arguments

`nodes`	Nodes in the admxture graph.
`parent_edges`	An $n \times 2$ matrix where the first column is the child node and the second is the parent.

Value

An incidence matrix for the child structure of an admixture graph.

See Also

agraph_parents

agraph_parents *Build the parent incidence matrix from an edge list.*

Description

Build the parent incidence matrix from an edge list.

Usage

```
agraph_parents(nodes, parent_edges)
```

Arguments

`nodes`	Nodes in the admxture graph.
`parent_edges`	An $n \times 2$ matrix where the first column is the child node and the second is the parent.

Value

An incidence matrix for the parent structure of an admixture graph.

See Also

`agraph_children`

`agraph_weights` *Build the matrix of admixture proportions from an edge list.*

Description

Build the matrix of admixture proportions from an edge list.

Usage

```
agraph_weights(nodes, admixture_weights, parents)
```

Arguments

`nodes`	The name of the nodes in the admxture graph.
`admixture_weights`	An $n \times 3$ matrix where the first column is the child node, the second isthe parent and the third is the admixture weight on that edge.
`parents`	The parent edge list. Used for checking graph consistency.

Value

A matrix containing the admixture weights.

`all_graphs` *All graphs.*

Description

Gives a list of all the graphs with at most a given number of admixture events. No duplicates.

Usage

```
all_graphs(populations, admixture_events)
```

Arguments

`populations`	A vector of populations (leaf names).
`admixture_events`	The maximum number of admixture events allowed.

Value

A list of admixture graphs.

See Also

`fit_graph_list`

`all_paths` *Compute all paths from one leaf to another.*

Description

Computes all paths from one leaf to another.

Usage

```
all_paths(graph, src, dst)
```

Arguments

`graph`	The admixture graph.
`src`	The starting leaf.
`dst`	The destination leaf.

Value

A list containing all the paths from `src` to `dst`.

`all_path_overlaps` *Get the list of overlaps of all paths.*

Description

Gets the list of overlaps of all paths.

Usage

```
all_path_overlaps(paths1, paths2)
```

Arguments

`paths1`	Paths between one pair of leaves.
`paths2`	Paths between another pair of leaves.

Value

A list of the overlaps of all combinations of paths from `paths1` and `paths2`.

bears *Statistics for populations of bears*

Description

Computed f_4(W,X;Y,Z) statistics for different populations of bears.

Usage

```
bears
```

Format

A data frame with 18 rows and 6 variables:

W The W population

X The X population

Y The Y population

Z The Z population

D The D (f_4(W,X;Y,Z)) statistics

Z.value The blocked jacknife Z values

Source

http://onlinelibrary.wiley.com/doi/10.1111/mec.13038/abstract

build_edge_optimisation_matrix *Build a matrix coding the linear system of edges once the admix variables have been fixed.*

Description

The elements are characters containing numbers, admix variable names, parenthesis and arithmetical operations. (Transform into expressions with `parse` and then evaluate with `eval`). The default column names are the edge names from `extract_graph_parameters`, the rows have no names.

Usage

```
build_edge_optimisation_matrix(data, graph,
  parameters = extract_graph_parameters(graph))
```

Arguments

`data`	The data set.
`graph`	The admixture graph.
`parameters`	In case one wants to tweak something in the graph.

Value

A list containing the full matrix (`full`), a version with zero columns removed (`column_reduced`) and parameters to pass forward (`parameters`).

`burn_in` *Removes the first k rows from a trace.*

Description

Removes the first k rows from a trace.

Usage

```
burn_in(trace, k)
```

Arguments

`trace`	A trace from an MCMC run.
`k`	Number of rows to discard as burn-in.

`calculate_concentration` *Building a proxy concentration matrix.*

Description

If we don't have the true concentration matrix of the data rows calculated, but at least have the Z scores of individual rows, (unrealistically) assuming independence and calculating the concentration matrix from those is still better than nothing (*i. e.* the identity matrix).

Usage

```
calculate_concentration(data, Z.value)
```

Arguments

`data`	The data containing at least the expected values of f statistics (column `D`) and possibly also products of expected values and f statistics divided by standard deviations of (the Z scores, column `Z.value`).
`Z.value`	Tells whether the Z scores are available or should we just use the identity matrix.

Value

The Cholesky decomposition of the inverted covariance matrix.

`canonise_expression` *Used to recognize similar expressions and to possibly simplify them.*

Description

It's best to simplify algebraic expression a little before evaluating.

Usage

```
canonise_expression(x)
```

Arguments

`x`	The input is assumed to be a `character` consisting of one or more terms. Each term starts with either `+` or `-` and after that contains one or more factors separated by `*`. Each factor is either an admix variable, a number or a clause `(1 - x)` (mind the spaces, this is how the function `f4` outputs), where `x` is again either an admix variable or a number. Everything is pretty much ruined if variable names are numbers or contain forbidden symbols `+, -, *, (, )`.

Value

A polynomial in a canonical form with no parenthesis or spaces and the monomials in lexicographical order. If everything is cancelled out then `+0`.

`canonise_graph` *Canonise graph.*

Description

Given a graph builds a unique logical vector depending only on the leaf names and the graph topology (not the inner node names, root position or the input order of edges or inner nodes). Can be used to detect graph isomorphism, that is, to weed out duplicates from a graph list. Only for comparison of graphs with the same leaf set!

Usage

```
canonise_graph(graph)
```

Arguments

`graph`	An admixture graph.

Value

A logical vector coding the parental matrix of a canonised version of the input graph.

`coef.agraph_fit` *Parameters for the fitted graph.*

Description

Extracts the graph parameters for the graph fitted to data. Note that the optimal parameters are generally not unique.

Usage

```
## S3 method for class 'agraph_fit'
coef(object, ...)
```

Arguments

`object`	The fitted object.
`...`	Additional parameters.

See Also

`link{summary.agraph_fit}`

`cost_function` *The cost function fed to Nelder-Mead.*

Description

We want Nelder-Mead to run fast so the cost function operates with the column reduced edge optimisation matrix and does not give any extra information about the fit. For the details, use `edge_optimisation_function` instead.

Usage

```
cost_function(data, concentration, matrix, graph,
  parameters = extract_graph_parameters(graph), iteration_multiplier = 3)
```

Arguments

`data`	The data set.
`concentration`	The Cholesky decomposition of the inverted covariance matrix.
`matrix`	A column reduced edge optimisation matrix (typically given by the function `build_edge_optimisation_matrix`).
`graph`	The admixture graph.
`parameters`	In case one wants to tweak something in the graph.
`iteration_multiplier`	Given to `mynonneg`.

Value

Given an input vector of admix variables, returns the smallest error regarding the edge variables.

See Also

`mynonneg`

`edge_optimisation_function`

`log_likelihood`

`edge` *Create an edge from a child to a parent.*

Description

Syntactic suggar for constructing edges in an admixture graph.

Usage

```
edge(child, parent)
```

Arguments

`child`	The name of the child node.
`parent`	The name of the parent node.

edge_optimisation_function

More detailed edge fitting than mere cost_function.

Description

Returning the cost, an example edge solution of an optimal fit, and linear relations describing the set of all edge solutions. Operating with the full edge optimisation matrix, not the column reduced one.

Usage

```
edge_optimisation_function(data, concentration, matrix, graph,
  parameters = extract_graph_parameters(graph), iteration_multiplier = 3)
```

Arguments

`data`	The data set.
`concentration`	The Cholesky decomposition of the inverted covariance matrix.
`matrix`	A full edge optimisation matrix (typically given by the function `build_edge_optimisation_matrix`).
`graph`	The admixture graph.
`parameters`	In case one wants to tweak something in the graph.
`iteration_multiplier`	Given to `mynonneg`.

Value

Given an input vector of admix variables, returns a list containing the minimal error (`cost`), the graph-$f4$ statistics (`approximation`), an example solution (`edge_fit`), linear relations describing all the solutions (`homogeneous`) and one way to choose the free (`free_edges`) and bounded (`bounded_edges`) edge variables.

See Also

`mynonneg`

`cost_function`

`log_likelihood`

eight_leaves_trees *Eight leaves trees.*

Description

Kind of obsolete since the introduction of all_graphs. A comprehensive listing of all the four unrooted trees with eight leaves. The position of the root can be moved later with the function make_an_outgroup.

Usage

```
eight_leaves_trees
```

Format

A list of functions on eight leaves. The outputs of these functions are agraph objects.

See Also

all_graphs

make_permutations

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

add_an_admixture2

make_an_outgroup

Other graphs: five_leaves_graphs, four_leaves_graphs, seven_leaves_graphs, six_leaves_graphs

Examples

```
# While the usage of this function is pretty self-explanatory, let's plot all the graphs
# just for browsing.
for (i in seq(1, length(eight_leaves_trees))) {
  graph <- eight_leaves_trees[[i]](c("A", "B", "C", "D", "E", "F", "G", "H"))
  # This is how you include quotation marks in strings by the way:
  title <- paste("eight_leaves_trees[[", i,
                 "]](c(\"A\", \"B\", \"C\", \"D\", \"E\", \"F\", \"G\", \"H\"))", sep = "")
  plot(graph, color = "brown", title = title)
}
```

`evaluate_f4` *Evaluates an f_4 statistics in a given environment.*

Description

Evaluates an f_4 statistics in a given environment.

Usage

```
evaluate_f4(graph, env, W, X, Y, Z)
```

Arguments

`graph`	The admixture graph.
`env`	The environment containing the graph parameters.
`W`	First population/sample.
`X`	Second population/sample.
`Y`	Third population/sample.
`Z`	Fourth population/sample.

Value

The f_4 value specified by the graph and the environment.

`examine_edge_optimisation_matrix` *Examine the edge optimisation matrix to detect unfitted admix variables.*

Description

If the essential number of equations is not higher than the essential number of edge variables, the quality of edge optimisation will not depend on the admix variables (expect possibly in isolated special cases where the quality can be worse), and a complaint will be given. Note: The admix variable not being fitted does not mean that there is no evidence of an admix event! Isolated values of the admix variables, possibly 0 or 1, might give significantly worse fit than a typical value (but not the other way around).

Usage

```
examine_edge_optimisation_matrix(matrix, tol = 1e-08)
```

Arguments

`matrix`	Not really a matrix but two (should be an output of `build_edge_optimisation_matrix`).
`tol`	Calulating the rank with `qr.solve` sometimes crashes. Default 10^{-8}.

Value

An indicator of warning (`complaint`), coding all the possibilities in a way that is interpreted elsewhere (in `summary.agraph_fit`).

See Also

`qr.solve`

`extract_admixture_proportion_parameters`

Extract the admixture proportion parameter from edge specifications.

Description

This function is simply selecting the edges with admixture proportion specifications so these can be handled when building a graph using `agraph`. It is not a function you would need to call explicitly, rather it is there to allow people *not* to use it to provide admixture proportions explicitly (which we normally wouldn't recommend).

Usage

```
extract_admixture_proportion_parameters(parent_edges)
```

Arguments

`parent_edges`	Matrix created with the `agraph_parents` function.

Value

The parents edges reduced to the rows with admixture proportions.

`extract_graph_parameters` *Extract all the parameters a graph contains.*

Description

The graph is parameterized by edge lengths and admixture proportions. This function extracts these parameters.

Usage

```
extract_graph_parameters(graph)
```

Arguments

`graph`	The admixture graph.

Value

A list containing two values: `edges`, a vector of edges and `admix_prop`, a vector containing admixture proportions.

`extract_trees` *Extract trees*

Description

Extracts all the trees embedded in an agraph object

Usage

```
extract_trees(graph)
```

Arguments

`graph`	An agraph object

Value

A list of trees

f2 *Calculate the f_2(A, B) statistics.*

Description

Calculate the $f_2(A, B)$ statistics.

Usage

```
f2(graph, A, B)
```

Arguments

graph	The admixture graph.
A	A leaf node.
B	A leaf node.

Value

A symbolic representation of the equation for the f_2 statistics given by the admixture graph.

f3 *Calculate the f_3(A; B, C) statistics.*

Description

Calculate the $f_3(A; B, C)$ statistics.

Usage

```
f3(graph, A, B, C)
```

Arguments

graph	The admixture graph.
A	A leaf node.
B	A leaf node.
C	A leaf node.

Value

A symbolic representation of the equation for the f_3 statistics given by the admixture graph.

f4 *Calculate the f_4(W, X; Y, Z) statistics.*

Description

Calculate the $f_4(W, X; Y, Z)$ statistics.

Usage

```
f4(graph, W, X, Y, Z)
```

Arguments

`graph`	The admixture graph.
`W`	A leaf node.
`X`	A leaf node.
`Y`	A leaf node.
`Z`	A leaf node.

Value

The overlaps between paths from `W` to `X` and paths from `Y` to `Z`.

f4stats *Make a data frame an f_4 statistics object.*

Description

This is mostly just a convinience function to set the class of a data frame such that we plot data as error bars in a meaningful way for statistics for admixture graphs.

Usage

```
f4stats(x)
```

Arguments

`x`	Data frame with observed D (f_4) statistics.

Value

Something about classes.

fast_fit *A fast version of graph fitting.*

Description

Given a table of observed f statistics and a graph, uses Nelder-Mead algorithm to find the graph parameters (edge lengths and admixture proportions) that minimize the value of cost_function, *i. e.* maximizes the likelihood of a graph with parameters given the observed data. Like fit_graph but dropping most of the analysis on the result. Intended for use in big iteration loops.

Usage

```
fast_fit(data, graph, point = list(rep(1e-05,
  length(extract_graph_parameters(graph)$admix_prop)), rep(1 - 1e-05,
  length(extract_graph_parameters(graph)$admix_prop))), Z.value = TRUE,
  concentration = calculate_concentration(data, Z.value),
  optimisation_options = NULL, parameters = extract_graph_parameters(graph),
  iteration_multiplier = 3)
```

Arguments

data	The data table, must contain columns W, X, Y, Z for sample names and D for the observed $f_4(W, X; Y, Z)$. May contain an optional column Z.value for the Z scores (the f statistics divided by the standard deviations).
graph	The admixture graph (an agraph object).
point	If the user wants to restrict the admixture proportions somehow, like to fix some of them. A list of two vectors: the lower and the upper bounds. As a default the bounds are just it little bit more than zero and less than one; this is because sometimes the infimum of the values of cost function is at a point of non-continuity, and zero and one have reasons to be problematic values in this respect.
Z.value	Whether we calculate the default concentration from Z scores (the default option TRUE) or just use the identity matrix.
concentration	The Cholesky decomposition of the inverted covariance matrix. Default matrix determined by the parameter Z.value.
optimisation_options	Options to the Nelder-Mead algorithm.
parameters	In case one wants to tweak something in the graph.
iteration_multiplier	Given to mynonneg.

Value

A list containing only the essentials about the fit: graph is the graph input, best_error is the minimal value of cost_function, obtained when the admixture proportions are best_fit.

See Also

cost_function

agraph

calculate_concentration

optimset

fit_graph

Examples

```
# For example, let's fit the following two admixture graph to an example data on bears:

data(bears)
print(bears)

leaves <- c("BLK", "PB", "Bar", "Chi1", "Chi2", "Adm1", "Adm2", "Denali", "Kenai", "Sweden")
inner_nodes <- c("R", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "M", "N")
edges <- parent_edges(c(edge("BLK", "R"),
                        edge("PB", "v"),
                        edge("Bar", "x"),
                        edge("Chi1", "y"),
                        edge("Chi2", "y"),
                        edge("Adm1", "z"),
                        edge("Adm2", "z"),
                        edge("Denali", "t"),
                        edge("Kenai", "s"),
                        edge("Sweden", "r"),
                        edge("q", "R"),
                        edge("r", "q"),
                        edge("s", "r"),
                        edge("t", "s"),
                        edge("u", "q"),
                        edge("v", "u"),
                        edge("w", "M"),
                        edge("x", "N"),
                        edge("y", "x"),
                        edge("z", "w"),
                        admixture_edge("M", "u", "t"),
                        admixture_edge("N", "v", "w")))
admixtures <- admixture_proportions(c(admix_props("M", "u", "t", "a"),
                                      admix_props("N", "v", "w", "b")))
bears_graph <- agraph(leaves, inner_nodes, edges, admixtures)
plot(bears_graph, show_admixture_labels = TRUE)

fit <- fast_fit(bears, bears_graph)
print(fit$best_error)

# The result is just the minimal value of the cost function and the values of admixture proportions
# where it's obtained, no deeper analysis of the fit.
```

fast_plot *Fast version of graph plotting.*

Description

This is a fast, deterministic and stand-alone function for visualizing the admixture graph. Has the bad habit if sometimes drawing several nodes at the exact same coordinates; for clearer reasults try `plot.agraph` (which, on the other hand, relies on numerical optimising of a compicated cost function and might be unpredictable).

Usage

```
fast_plot(x, ordered_leaves = NULL, show_admixture_labels = FALSE,
  show_inner_node_labels = FALSE, ...)
```

Arguments

`x`	The admixture graph.
`ordered_leaves`	The leaf-nodes in the left to right order they should be drawn.
`show_admixture_labels`	A flag determining if the plot should include the names of admixture proportions.
`show_inner_node_labels`	A flag determining if the plot should include the names of inner nodes.
`...`	Additional plotting options.

Value

A plot.

See Also

`plot.agraph`

Examples

```
# taken from the collection of all the admixture graphs with four leaves and at
# most two admixture events:

fast_plot(four_leaves_graphs[[24]](c("A", "B", "C", "D")))

# To be fair, here is a graph that looks all right:

fast_plot(four_leaves_graphs[[25]](c("A", "B", "C", "D")))
```

`filter_on_leaves` *Filter data so all W, X, Y and Z are leaves in the graph.*

Description

Filter data so all `W`, `X`, `Y` and `Z` are leaves in the graph.

Usage

```
filter_on_leaves(data, graph)
```

Arguments

`data`	Data frame (or similar) object containing columns `W`, `X`, `Y`, and `Z`.
`graph`	Admixture graph.

Value

Data frame with rows where `W`, `X`, `Y`, or `Z` are not leaves are removed.

`fitted.agraph_fit` *Predicted f statistics for the fitted graph.*

Description

Gets the predicted f statistics F for the fitted graph.

Usage

```
## S3 method for class 'agraph_fit'
fitted(object, ...)
```

Arguments

`object`	The fitted object.
`...`	Additional parameters.

See Also

`link{summary.agraph_fit}`

fit_graph *Fit the graph parameters to a data set.*

Description

Given a table of observed f statistics and a graph, uses Nelder-Mead algorithm to find the graph parameters (edge lengths and admixture proportions) that minimize the value of `cost_function`, *i. e.* maximizes the likelihood of a graph with parameters given the observed data. Like `fast_fit` but outputs a more detailed analysis on the results.

Usage

```
fit_graph(data, graph, point = list(rep(1e-05,
  length(extract_graph_parameters(graph)$admix_prop)), rep(1 - 1e-05,
  length(extract_graph_parameters(graph)$admix_prop))), Z.value = TRUE,
  concentration = calculate_concentration(data, Z.value),
  optimisation_options = NULL, parameters = extract_graph_parameters(graph),
  iteration_multiplier = 3, qr_tol = 1e-08)
```

Arguments

`data`	The data table, must contain columns `W`, `X`, `Y`, `Z` for sample names and `D` for the observed $f_4(W, X; Y, Z)$. May contain an optional column `Z.value` for the Z scores (the f statistics divided by the standard deviations).
`graph`	The admixture graph (an `agraph` object).
`point`	If the user wants to restrict the admixture proportions somehow, like to fix some of them. A list of two vectors: the lower and the upper bounds. As a default the bounds are just it little bit more than zero and less than one; this is because sometimes the infimum of the values of cost function is at a point of non-continuity, and zero and one have reasons to be problematic values in this respect.
`Z.value`	Whether we calculate the default concentration from Z scores (the default option `TRUE`) or just use the identity matrix.
`concentration`	The Cholesky decomposition of the inverted covariance matrix. Default matrix determined by the parameter `Z.value`.
`optimisation_options`	Options to the Nelder-Mead algorithm.
`parameters`	In case one wants to tweak something in the graph.
`iteration_multiplier`	Given to `mynonneg`.
`qr_tol`	Given to `examine_edge_optimisation_matrix`.

Value

A class agraph_fit list containing a lot of information about the fit:
data is the input data,
graph is the input graph,
matrix is the output of build_edge_optimisation_matrix, containing the full matrix, the column_reduced matrix without zero columns, and graph parameters,
complaint coding wchich subsets of admixture proportions are turly fitted,
best_fit is the optimal admixture proportions (might not be unique if they are not turly fitted),
best_edge_fit is an example of optimal edge lengths,
homogeneous is the reduced row echelon form of the matrix describing when a vector of edge lengths have no effect on the prediced statistics F,
free_edges is one way to choose a subset of edge lengths in such a vector as free variables,
bounded_edges is how we calculate the reamining edge lengths from the free ones,
best_error is the minimum value of the cost_function,
approximation is the predicted statistics F with the optimal graph parameters,
parameters is jsut a shortcut for the graph parameters.
See summary.agraph_fit for the interpretation of some of these results.

See Also

cost_function

agraph

calculate_concentration

optimset

fast_fit

Examples

```
# For example, let's fit the following two admixture graph to an example data on bears:

data(bears)
print(bears)

leaves <- c("BLK", "PB", "Bar", "Chi1", "Chi2", "Adm1", "Adm2", "Denali", "Kenai", "Sweden")
inner_nodes <- c("R", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "M", "N")
edges <- parent_edges(c(edge("BLK", "R"),
                        edge("PB", "v"),
                        edge("Bar", "x"),
                        edge("Chi1", "y"),
                        edge("Chi2", "y"),
                        edge("Adm1", "z"),
                        edge("Adm2", "z"),
                        edge("Denali", "t"),
                        edge("Kenai", "s"),
                        edge("Sweden", "r"),
                        edge("q", "R"),
                        edge("r", "q"),
                        edge("s", "r"),
                        edge("t", "s"),
                        edge("u", "q"),
                        edge("v", "u"),
                        edge("w", "M"),
                        edge("x", "N"),
                        edge("y", "x"),
                        edge("z", "w"),
                        admixture_edge("M", "u", "t"),
                        admixture_edge("N", "v", "w")))
admixtures <- admixture_proportions(c(admix_props("M", "u", "t", "a"),
                                      admix_props("N", "v", "w", "b")))
bears_graph <- agraph(leaves, inner_nodes, edges, admixtures)
plot(bears_graph, show_admixture_labels = TRUE)

fit <- fit_graph(bears, bears_graph)
summary(fit)

# It turned out the values of admixture proportions had no effect on the cost function. This is not
# too surprising because the huge graph contains a lot of edge variables compared to the tiny
# amount of data we used! Note however that the mere existence of the admixture event with non-
# trivial (not zero or one) admixture proportion might still decrease the cost function.
```

`fit_graph_list` *Fit lots of graphs to data.*

Description

Fits a list of graphs to given data using parallel computation. This function needs `doParallel`, `foreach` and `parallel` installed.

Usage

```
fit_graph_list(data, graphs, cores)
```

Arguments

`data`	The data table.
`graphs`	List of graphs.
`cores`	Number of cores used.

Value

A list of `fast_fit` results.

See Also

`all_graphs`

`fit_permutations_and_graphs`

`fit_permutations_and_graphs` *Fit lots of graphs to data.*

Description

Combines a list of (population) permutations and a list of graph topologies to a big list of graphs, then fits those graphs to given data using parallel computation. This function needs `doParallel`, `foreach` and `parallel` installed.

Usage

```
fit_permutations_and_graphs(data, permutations, graphs, cores)
```

Arguments

`data`	The data table.
`permutations`	List of population permutations.
`graphs`	List of functions for producing graphs.
`cores`	Number of cores used.

Value

A list of `fast_fit` results.

See Also

`make_permutations`

`four_leaves_graphs`

`five_leaves_graphs`

`six_leaves_graphs`

`seven_leaves_graphs`

`eight_leaves_trees`

`fit_graph_list`

Examples

```
# Let's experiment by fitting all the graphs with five leaves and at most one admixture
# event to a five population subset of the bear data. Note that with three data rows only
# we do wisely by not concluding too much about the actual bear family tree; this is to
# illustrate the function usage only!

data(bears)
data <- bears[16:18, ]
```

```
print(data)
permutations <- make_permutations(c("PB", "BLK", "Sweden", "Denali", "Kenai"))
graphs <- five_leaves_graphs

# We go with one core only as I don't know what kind of machine you are using.

fitted_graphs <- fit_permutations_and_graphs(data, permutations, graphs, 1)

# Now sort the fitted objects by best_error and see how the best graph looks like.

errors <- sapply(fitted_graphs, function(x) x$best_error)
best_graphs <- fitted_graphs[order(errors)]
plot(best_graphs[[1]]$graph, color = "goldenrod", title = best_graphs[[1]]$best_error)

# The same value for best_error actually occurs in the list 152 times because of our
# unsufficient data.
```

five_leaves_graphs *Five leaves graphs.*

Description

Kind of obsolete since the introduction of all_graphs. A comprehensive listing of all the 132 admixture graphs with five leaves and at most two admixture events. Our convention is that the position of the root does not matter (as long as it's not after an admixture event) and that graphs that have *eyes*, two inner nodes with the property that all the paths between any two leaves visits both or neither of them, are excluded. The reason is that the f statistics can't detect the exact position of the root or distinguish between an eye and a simple branch. The position of the root can be moved later with the function make_an_outgroup.

Usage

```
five_leaves_graphs
```

Format

A list of functions on five leaves and a parameter permutations which is FALSE by default. The outputs of these functions are either single agraph objects with the input vector as leaves, or if permutations is TRUE, lists of all the possible agraph objects with that leaf set up to symmetry.

See Also

all_graphs

make_permutations

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

add_an_admixture2

make_an_outgroup

Other graphs: eight_leaves_trees, four_leaves_graphs, seven_leaves_graphs, six_leaves_graphs

Examples

```
# While the usage of this function is pretty self-explanatory, let's plot all the graphs
# just for browsing.
for (i in seq(1, length(five_leaves_graphs))) {
  graph <- five_leaves_graphs[[i]](c("A", "B", "C", "D", "E"))
  # This is how you include quotation marks in strings by the way:
  title <- paste("five_leaves_graphs[[", i, "]](c(\"A\", \"B\", \"C\", \"D\", \"E\"))",
                 sep = "")
  plot(graph, color = "purple", title = title)
}
```

---

format_path *Create a path data frame from a list of nodes.*

Description

Creates a path data frame from a list of nodes.

Usage

```
format_path(graph, nodes)
```

Arguments

graph	The admixture graph the path is in.
nodes	A list of nodes on a path.

Value

A data frame capturing the path and the probabilities/weights on the edges.

four_leaves_graphs *Four leaves graphs.*

Description

Kind of obsolete since the introduction of all_graphs. A comprehensive listing of all the 37 admixture graphs with four leaves and at most two admixture events. Our convention is that the position of the root does not matter (as long as it's not after an admixture event) and that graphs that have *eyes*, two inner nodes with the property that all the paths between any two leaves visits both or neither of them, are excluded. The reason is that the f statistics can't detect the exact position of the root or distinguish between an eye and a simple branch. The position of the root can be moved later with the function make_an_outgroup.

Usage

```
four_leaves_graphs
```

Format

A list of functions on four leaves and a parameter `permutations` which is `FALSE` by default. The outputs of these functions are either single agraph objects with the input vector as leaves, or if `permutations` is `TRUE`, lists of all the possible agraph objects with that leaf set up to symmetry.

See Also

all_graphs

make_permutations

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

add_an_admixture2

make_an_outgroup

Other graphs: eight_leaves_trees, five_leaves_graphs, seven_leaves_graphs, six_leaves_graphs

Examples

```
# While the usage of this function is pretty self-explanatory, let's plot all the graphs
# just for browsing.
for (i in seq(1, length(four_leaves_graphs))) {
  graph <- four_leaves_graphs[[i]](c("A", "B", "C", "D"))
  # This is how you include quotation marks in strings by the way:
  title <- paste("four_leaves_graphs[[", i, "]](c(\"A\", \"B\", \"C\", \"D\"))", sep = "")
  plot(graph, color = "tomato3", title = title)
}
```

get_graph_f4_sign *Extracts the sign for the f_4 statistics predicted by the graph.*

Description

Extracts the sign for the f_4 statistics predicted by the graph.

Usage

```
get_graph_f4_sign(graph, W, X, Y, Z)
```

Arguments

`graph`	The admixture graph.
`W`	First population/sample.
`X`	Second population/sample.
`Y`	Third population/sample.
`Z`	Fourth population/sample.

Value

The sign of the f_4 specified by the graph (or `NA` when it cannot be determined without knowing the graph parameters).

graphs_2_0 *Admixture graphs of 2 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_2_0
```

Format

A data frame with 1 row and 9 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_3_0` *Admixture graphs of 3 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_3_0
```

Format

A data frame with 1 row and 25 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_3_1` *Admixture graphs of 3 leaves and 1 admixture event compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_3_1
```

Format

A data frame with 3 rows and 49 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_4_0` *Admixture graphs of 4 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_4_0
```

Format

A data frame with 3 rows and 49 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_4_1` *Admixture graphs of 4 leaves and 1 admixture event compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_4_1
```

Format

A data frame with 30 rows and 81 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_4_2` *Admixture graphs of 4 leaves and 2 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_4_2
```

Format

A data frame with 486 rows and 121 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_5_0` *Admixture graphs of 5 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_5_0
```

Format

A data frame with 15 rows and 81 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_5_1` *Admixture graphs of 5 leaves and 1 admixture event compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_5_1
```

Format

A data frame with 315 rows and 121 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_5_2` *Admixture graphs of 5 leaves and 2 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_5_2
```

Format

A data frame with 7710 rows and 169 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

graphs_6_0 — *Admixture graphs of 6 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_6_0
```

Format

A data frame with 105 rows and 121 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

graphs_6_1 — *Admixture graphs of 6 leaves and 1 admixture event compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_6_1
```

Format

A data frame with 3780 rows and 169 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_6_2` *Admixture graphs of 6 leaves and 2 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_6_2
```

Format

A data frame with 131400 rows and 225 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_7_0` *Admixture graphs of 7 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_7_0
```

Format

A data frame with 945 rows and 169 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_7_1` *Admixture graphs of 7 leaves and 1 admixture event compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_7_1
```

Format

A data frame with 51975 rows and 225 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

`graphs_8_0` *Admixture graphs of 8 leaves and 0 admixture events compressed into vectors*

Description

Use `vector_to_graph` on the `n`:th row to access the `n`:th graph as an `agraph` object.

Usage

```
graphs_8_0
```

Format

A data frame with 10395 rows and 225 variables.

Source

Calculated using `all_graphs` and `graph_to_vector`.

graph_environment — *Build an environment in which f statistics can be evaluated.*

Description

Constructs an environment in which the f statistics for a graph can be evaluted, based on the parameters in a graph and values for edge lengths and admixture proportions (with defaults if not specified).

Usage

```
graph_environment(parameters, edge_lengths = NULL, admix_prop = NULL)
```

Arguments

`parameters`	The parameters of a graph as returned by `extract_graph_parameters`.
`edge_lengths`	If specified, a vector of edge lengths. Otherwise defaults are used.
`admix_prop`	If specified, a vector of admixture proportions. Otherwise defaults are used.

Value

A list containing two values: `edges`, a vector of edges and `admix_prop`, a vector containing admixture proportions.

graph_to_vector — *Graph to vector.*

Description

Encodes an `agraph` object into a logical vector for saving memory. The admixture proportion names will be lost.

Usage

```
graph_to_vector(graph)
```

Arguments

`graph`	A graph.

Value

The logical vector representing the graph.

`is_descendant_of` *Is descendant of.*

Description

Tells whether a given node is descendant of another given node in a graph.

Usage

```
is_descendant_of(graph, offspring, ancestor)
```

Arguments

`graph`	A graph.
`offspring`	Potential offspring.
`ancestor`	Potential ancestor.

Value

A truth value.

`is_negative` *All overlaps are either empty or have a negative weight.*

Description

All overlaps are either empty or have a negative weight.

Usage

```
is_negative(overlaps)
```

Arguments

`overlaps`	Data frame representing path overlaps, typically generated by `all_path_overlaps`.

`is_positive` *All overlaps are either empty or have a positive weight.*

Description

All overlaps are either empty or have a positive weight.

Usage

```
is_positive(overlaps)
```

Arguments

`overlaps`	Data frame representing path overlaps, typically generated by `all_path_overlaps`.

`is_unknown` *Overlapping edges have both positive and negative contributions.*

Description

Overlapping edges have both positive and negative contributions.

Usage

```
is_unknown(overlaps)
```

Arguments

`overlaps`	Data frame representing path overlaps, typically generated by `all_path_overlaps`.

`is_zero` *All overlaps are empty.*

Description

All overlaps are empty.

Usage

```
is_zero(overlaps)
```

Arguments

`overlaps`	Data frame representing path overlaps, typically generated by `all_path_overlaps`.

log_likelihood

Calculate (essentially) the log likelihood of a graph with parameters, given the observation.

Description

Or the log likelihood of the observation, given graph with parameters, depending how things are modeled. Basically this is just cost_function that doesn't optimize the edge variables but has them as an argument instead.

Usage

```
log_likelihood(f, concentration, matrix, graph,
  parameters = extract_graph_parameters(graph))
```

Arguments

f	The observed f statistics (the column D from data).
concentration	The Cholesky decomposition of the inverted covariance matrix. So if S is the covariance matrix, this is $C = chol(S^{-1})$ satisfying $S^{-1} = C^t C$.
matrix	A column reduced edge optimisation matrix (typically given by the function build_edge_optimisation_matrix).
graph	The admixture graph. Here to give default value for:
parameters	Just because we need to know variable names.

Value

The output is a function. Given admixture proportions x and edge lengths e, the graph topology T implies an estimate F for the statistics f. This output function calculates

$$l = (F - f)^t S^{-1} (F - f)$$

from x and e. Up to a constant error and multiplier that is a log likelihood function, as

$$\det(2\pi S)^{-1/2} e^{-l/2}$$

can be seen as a likelihood of a graph with parameters, given the observation, or the other way around (possibly up to a normalization constant).

See Also

cost_function

edge_optimisation_function

calculate_concentration

`log_sum_of_logs` *Computes the log of a sum of numbers all given in log-space.*

Description

Given a sequence of numbers $[\log(x_1), \log(x_2), ..., \log(x_n)]$, computes $\log(\sum_{i=1}^{n} x_i)$. For adding two numbers that are given in log space we use the expression `max(x, y) + log1p(exp(-abs(x - y)))` which is a good approximation if `x` and `y` are of the same order of magnitude, but if they are of very different sizes just returns the maximum of the two. To prevent adding numbers of very different magnitude we iteratively add the numbers pairwise. Because of numerical issues with doing this, the order of the input values can affect the result.

Usage

```
log_sum_of_logs(log_values)
```

Arguments

`log_values`	Sequence of numbers in log space $[\log(x_1), \log(x_2), ..., \log(x_n)]$.

Value

$\log(\sum_{i=1}^{n} x_i)$.

`make_an_outgroup` *Make an outgroup.*

Description

Given a graph and a leaf, tries to put the root of the graph on the edge leading to the leaf. If not possible (*i. e.* if the leaf has admixture in its ancestry), puts the root somewhere else.

Usage

```
make_an_outgroup(graph, outgroup = "", all_neutral = FALSE)
```

Arguments

`graph`	An admixture graph.
`outgroup`	A leaf we want to be the outgroup.
`all_neutral`	For when other functions need to root graphs in a neutral way.

Value

An admixture graph with the given leaf as an outgroup, if possible.

See Also

make_permutations

four_leaves_graphs

five_leaves_graphs

six_leaves_graphs

seven_leaves_graphs

eight_leaves_trees

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

add_an_admixture2

Examples

```
# Here is a little family tree of some dinosaur-like animals.

species <- c("triceratops", "crocodile", "diplodocus", "tyrannosaurus", "chicken")
graph <- five_leaves_graphs[[1]](species)
plot(graph)

# Of course we know that while this is correct as an undirected graph, "crocodile"
# should really be the outgroup.

graph <- make_an_outgroup(graph, "crocodile")
plot(graph)

# Strictly speaking the graph is still a little misleading because unfortunately
# the (non-bird) dinosaurs are extinct :-(
```

---

make_mcmc_model *Collect the information about a graph and a data set needed to run an MCMC on it.*

Description

Collect the information about a graph and a data set needed to run an MCMC on it.

Usage

```
make_mcmc_model(graph, data)
```

Arguments

`graph`	The admixture graph to analyse.
`data`	The data set to compute the posterior over.

Value

A model object wrapping functions and data needed to sample from the MCMC.

`make_permutations` *List of permutations.*

Description

List of permutations of given elements.

Usage

```
make_permutations(populations)
```

Arguments

`populations`	A vector (of populations for example) of length between 4 and 8.

Value

A list of different permutations of the elements of `x`.

See Also

`four_leaves_graphs`

`five_leaves_graphs`

`six_leaves_graphs`

`seven_leaves_trees`

`eight_leaves_trees`

`fit_permutations_and_graphs`

`add_a_leaf`

`add_an_admixture`

`add_an_admixture2`

Examples

```
# The number of permutations of n elements is n!. Take 0! = 1, 1! = 1, 2! = 2
# and 3! = 6 for granted. Now we can estimate e:
FOUR <- length(make_permutations(c(1, 2, 3, 4)))
FIVE <- length(make_permutations(c(1, 2, 3, 4, 5)))
SIX <- length(make_permutations(c(1, 2, 3, 4, 5, 6)))
SEVEN <- length(make_permutations(c(1, 2, 3, 4, 5, 6, 7)))
EIGHT <- length(make_permutations(c(1, 2, 3, 4, 5, 6, 7, 8)))
1/1 + 1/1 + 1/2 + 1/6 + 1/FOUR + 1/FIVE + 1/SIX + 1/SEVEN + 1/EIGHT
# Hey that was pretty close!
```

`model_bayes_factor_n` *Computes the Bayes factor between two models from samples from their posterior distributions.*

Description

The likelihood of a graph can be computed by integrating over all the graph parameters (with appropriate priors). Doing this by sampling from priors is very inefficient, so we use samples from the posteriors to importance sample the likelihood. Given two graphs, and samples from their posteriors, we can estimate the Bayes factor between them.

Usage

```
model_bayes_factor_n(logL1, logL2, no_samples = 100)
```

Arguments

`logL1`	Samples of log likelihoods from the posterior distribution of the first graph.
`logL2`	Samples of log likelihoods from the posterior distribution of the second graph.
`no_samples`	Number of permutations to sample when computing the result.

Details

The numerical issues with adding a lot of numbers in log space is unstable so we get a better estimate by doing it several times on different permutations of the data. This function calculates the mean of the Bayes factors over different permutations of the input and estimates the standard deviation.

Value

The Bayes factor between the two graphs given as the mean and standard deviation over `no_samples` different permutations of the input.

`model_likelihood` *Computes the likelihood of a model from samples from its posterior distribution.*

Description

The likelihood of a graph can be computed by integrating over all the graph parameters (with appropriate priors). Doing this by sampling from priors is very inefficient, so we use samples from the posteriors to importance sample the likelihood.

Usage

```
model_likelihood(log_likelihoods)
```

Arguments

`log_likelihoods`	Samples of log likelihoods from the posterior distribution of the graph.

Value

The likelihood of a graph where graph parameters are integrated out.

`model_likelihood_n` *Computes the likelihood of a model from samples from its posterior distribution.*

Description

The likelihood of a graph can be computed by integrating over all the graph parameters (with appropriate priors). Doing this by sampling from priors is very inefficient, so we use samples from the posteriors to importance sample the likelihood.

Usage

```
model_likelihood_n(log_likelihoods, no_samples = 100)
```

Arguments

`log_likelihoods`	Samples of log likelihoods from the posterior distribution of the graph.
`no_samples`	Number of permutations to sample when computing the result.

Details

The numerical issues with adding a lot of numbers in log space is unstable so we get a better estimate by doing it several times on different permutations of the data.This function calculates the mean of the likelihoods over different permutations of the input and estimates the standard devition.

Value

The likelihood of a graph where graph parameters are integrated out given as the mean and standard deviation over `no_samples` different permutations of the input.

`mynonneg` *Non negative least square solution.*

Description

This is the function `lsqnonneg` from the package `pracma`, I just changed `qr.solve` into using Moore-Penrose inverse instead (`ginv` from `MASS`) as `qr.solve` crashes for some singular inputs. Now it won't crash but it's sometimes running for very long time (forever?), presumably with those problematic inputs. After too many steps the function halts and lies that the fit was terrible. I don't think this will cause problems.

Usage

```
mynonneg(C, d, iteration_multiplier = 3)
```

Arguments

`C`	The matrix.
`d`	The vector.
`iteration_multiplier`	The definition of "too many steps". Default value is 3 (times 10 times the matrix height).

Value

A vector (`x`) and the error (`resid.norm`).

See Also

`lsqnonneg`

`qr.solve`

`ginv`

`no_admixture_events` *Get the number of admixture events in a graph.*

Description

Get the number of admixture events in a graph.

Usage

```
no_admixture_events(x)
```

Arguments

`x`	The graph.

Value

Number of admixture events in the graph.

`no_admixture_events.agraph` *Get the number of admixture events in a graph.*

Description

Get the number of admixture events in a graph.

Usage

```
## S3 method for class 'agraph'
no_admixture_events(x)
```

Arguments

`x`	The graph.

Value

Number of admixture events in the graph.

no_admixture_events.agraph_fit

Get the number of admixture events in a fitted graph.

Description

Get the number of admixture events in a fitted graph.

Usage

```
## S3 method for class 'agraph_fit'
no_admixture_events(x)
```

Arguments

x	The fitted graph.

Value

Number of admixture events in the graph.

no_admixture_events.agraph_fit_list

Get the number of admixture events in a list of fitted graph.

Description

Get the number of admixture events in a list of fitted graph.

Usage

```
## S3 method for class 'agraph_fit_list'
no_admixture_events(x)
```

Arguments

x	The graphs.

Value

Number of admixture events in the graphs.

`no_poor_fits` *Get the number of tests in the fit where the predictions fall outside of the error bars.*

Description

Get the number of tests in the fit where the predictions fall outside of the error bars.

Usage

```
no_poor_fits(fit, sigma = 6)
```

Arguments

`fit`	The fitted graph.
`sigma`	The width of the error bars.

Value

The poorly fitted tests.

`no_poor_fits.agraph_fit` *Get the number of tests in the fit where the predictions fall outside of the error bars.*

Description

Get the number of tests in the fit where the predictions fall outside of the error bars.

Usage

```
## S3 method for class 'agraph_fit'
no_poor_fits(fit, sigma = 6)
```

Arguments

`fit`	The fitted graph.
`sigma`	The width of the error bars.

Value

The poorly fitted tests.

`no_poor_fits.agraph_fit_list` *Get the number of tests in the fit where the predictions fall outside of the error bars.*

Description

Get the number of tests in the fit where the predictions fall outside of the error bars.

Usage

```
## S3 method for class 'agraph_fit_list'
no_poor_fits(fit, sigma = 6)
```

Arguments

`fit`	The fitted graph.
`sigma`	The width of the error bars.

Value

The poorly fitted tests.

`overlaps_sign` *Get the sign of overlapping paths.*

Description

Get the sign of overlapping paths.

Usage

```
overlaps_sign(overlaps)
```

Arguments

`overlaps`	Data frame representing path overlaps, typically generated by `all_path_overlaps`.

`parent_edges` *Create the list of edges for an admixture graph.*

Description

Syntactic suggar for constructing edges in an admixture graph.

Usage

```
parent_edges(edges)
```

Arguments

`edges`	List of edges.

`path_overlap` *Collect the postive and negative overlap between two paths.*

Description

Collects the postive and negative overlap between two paths.

Usage

```
path_overlap(path1, path2)
```

Arguments

`path1`	The first path.
`path2`	The second path.

Value

The (admixture) probability of seeing the two paths together with the positive and negative edges in the overlap.

plot.agraph *Plot an admixture graph.*

Description

This is a basic drawing routine for visualising the graph. Uses Nelder-Mead algorithm and complicated heuristic approach to find aestethic node coordinates, but due to bad luck or an oversight in the heuristics, especially with larger graphs, might sometimes produce a weird looking result. Usually plotting again helps and if not, use the optional parameters to help the algorithm or try the faster and deterministic function `fast_plot` (which unfortunately is not very good at handling multiple admixture events).

Usage

```
## S3 method for class 'agraph'
plot(x, show_leaf_labels = TRUE, draw_leaves = TRUE,
  color = "yellowgreen", show_inner_node_labels = FALSE,
  draw_inner_nodes = FALSE, inner_node_color = color,
  show_admixture_labels = FALSE, parent_order = list(),
  child_order = list(), leaf_order = NULL, fix = list(), platform = 1,
  title = NULL, ...)
```

Arguments

`x`	The admixture graph.
`show_leaf_labels`	A flag determining if leaf names are shown.
`draw_leaves`	A flag determining if leaf nodes are drawn as little circles.
`color`	Color of all drawn nodes unless overriden by `inner_node_color`.
`show_inner_node_labels`	A flag determining if the plot should include the names of inner nodes.
`draw_inner_nodes`	A flag determining if inner nodes are drawn as little circles.
`inner_node_color`	Color of inner node circles, if drawn.
`show_admixture_labels`	A flag determining if the plot should include the names of admixture proportions.
`parent_order`	An optional list of instuctions on which order from left to right to draw the parents of nodes. The list should contain character vectors of parents with the name of the child, *e.g.* `child = c("left_parent", "right_parent")`. Using automated heuristics for nodes not specified.
`child_order`	An optional list of instuctions on which order from left to right to draw the children of nodes. The list should contain character vectors of children with the name of the parent, *e.g.* `parent = c("left_child", "right_child")`. Using automated heuristics for nodes not specified.

`leaf_order`	An optional vector describing in which order should the leaves be drawn. Using automated heuristic depending on `parent_order` and `child_order` if not specified. Accepts both a character vector of the leaves or a numeric vector interpreted as a permutation of the default order.
`fix`	If nothing else helps, the list `fix` can be used to correct the inner node coordinates given by the heuristics. Should contain numeric vectors of length 2 with the name of an inner node, *e.g.* `inner_node = c(0, 10)`, moving `inner_node` to the right 10 units where 100 is the plot width. Non-specified inner nodes are left in peace.
`platform`	By default admixture nodes are drawn with a horizontal platform for proportion labels, the width of which is half the distance between any two leaves. The number `platform` tells how many default platform widths should the platforms be wide, *i. e.* zero means no platform.
`title`	Optional title for the plot.
`...`	Additional plotting options.

Value

A plot.

See Also

agraph

fast_plot

Examples

```
leaves <- c("salmon", "sea horse", "mermaid", "horse", "human", "lobster")
inner_nodes <- c("R", "s", "t", "u", "v", "w", "x", "y", "z")
edges <- parent_edges(c(edge("salmon", "t"),
                        edge("sea horse", "y"),
                        edge("mermaid", "z"),
                        edge("horse", "w"),
                        edge("human", "x"),
                        edge("lobster", "R"),
                        edge("s", "R"),
                        edge("t", "s"),
                        edge("u", "t"),
                        edge("v", "s"),
                        edge("w", "v"),
                        edge("x", "v"),
                        admixture_edge("y", "u", "w"),
                        admixture_edge("z", "u", "x")))
admixtures <- admixture_proportions(c(admix_props("y", "u", "w", "a"),
                                      admix_props("z", "u", "x", "b")))
graph <- agraph(leaves, inner_nodes, edges, admixtures)
plot(graph, show_inner_node_labels = TRUE)
```

```
# Suppose now that we prefer to have the outgroup "lobster" on the right side.
# This is achieved by telling the algorithm that the children of "R" should be in
# the order ("s", "lobster"), from left to right:

plot(graph, child_order = list(R = c("s", "lobster")))

# Suppose further that we prefer to have "mermaid" and "human" next to each other,
# as well as "sea horse" and "horse". This is easily achieved by rearranging the leaf
# order proposed by the algorithm. We can also fine-tune by moving "y" a little bit
# to the right, make the admixture platforms a bit shorter, color the nodes, show the
# admixture proportions and give the plot a title:

plot(graph, child_order = list(R = c("s", "lobster")), leaf_order = c(1, 2, 4, 3, 5, 6),
     fix = list(y = c(5, 0)), platform = 0.8, color = "deepskyblue",
     inner_node_color = "skyblue", show_admixture_labels = TRUE,
     title = "Evolution of fish/mammal hybrids")
```

plot.agraph_fit *Plot the fit of a graph to data.*

Description

Plot the fit of a graph to data.

Usage

```
## S3 method for class 'agraph_fit'
plot(x, sigma = 6, grayscale = FALSE, ...)
```

Arguments

x	Fitted graph object.
sigma	How many standard deviations the error bars should be wide.
grayscale	Should the plot be in black and white?
...	Additional parameters.

plot.f4stats — *Plot the fit of a graph to data.*

Description

Plot the fit of a graph to data.

Usage

```
## S3 method for class 'f4stats'
plot(x, sigma = 6, ...)
```

Arguments

`x`	Data frame with observed D (f_4) statistics
`sigma`	How many sigmas the error bars should be wide.
`...`	Additional parameters.

plot_fit_1 — *A plot of the cost function or number of fitted statistics.*

Description

A plot of the cost function with respect to one admix variable specified by the user. Sorry about the name, all the good ones were taken and the fact that the word "graph" means two different things doesn't help any.

Usage

```
plot_fit_1(object, X, resolution = 100, show_fit = FALSE, sigma = 6, ...)
```

Arguments

`object`	The fitted object.
`X`	An admix variable name (remember quotation marks).
`resolution`	How densely is the function evaluated.
`show_fit`	Should the function plot the number of statistics where the graph fits the data instead of the `cost_function`?
`sigma`	If `show_fit` is `TRUE` then each statistic is considered fitted if the difference between a prediction and the observation statistics is no more than $D * \sigma/(2 * Z)$. Notice that even when plotting the number of fitted statistics, we have no guarantee that the chosen variables maximize this number as the fitting function still optimizes `cost_function`.
`...`	Additional parameters.

Value

Values for optimal cost function for values of `X` between zero and one, plotted.

See Also

`plot_fit_2`

`plot_fit_2` *A contour plot of the cost function.*

Description

A contour plot of the cost function with respect to two admix variables specified by the user.

Usage

```
plot_fit_2(object, X, Y, resolution = 10, show_fit = FALSE, sigma = 6,
  grayscale = FALSE, ...)
```

Arguments

`object`	The fitted object.
`X`	An admix variable name (remember quotation marks).
`Y`	An admix variable name (remember quotation marks).
`resolution`	How densely is the function evaluated.
`show_fit`	Should the function plot the number of statistics where the graph fits the data instead of the `cost_function`?
`sigma`	If `show_fit` is `TRUE` then each statistic is considered fitted if the difference between a prediction and the observation statistics is no more than $D * \sigma/(2 * Z)$. Notice that even when plotting the number of fitted statistics, we have no guarantee that the chosen variables maximize this number as the fitting function still optimizes `cost_function`.
`grayscale`	Should the figure be plotted in grayscale or in colour?
`...`	Additional parameters passed to the plotting function `contour`.

Value

The matrix of values computed and plotted.

See Also

`contour`

`plot_fit_1`

poor_fits — *Get the tests in the fit where the predictions fall outside of the error bars.*

Description

Get the tests in the fit where the predictions fall outside of the error bars.

Usage

```
poor_fits(fit, sigma = 6)
```

Arguments

`fit`	The fitted graph.
`sigma`	The width of the error bars.

Value

The poorly fitted tests.

poor_fits.agraph_fit — *Get the tests in the fit where the predictions fall outside of the error bars.*

Description

Get the tests in the fit where the predictions fall outside of the error bars.

Usage

```
## S3 method for class 'agraph_fit'
poor_fits(fit, sigma = 6)
```

Arguments

`fit`	The fitted graph.
`sigma`	The width of the error bars.

Value

The poorly fitted tests.

poor_fits.agraph_fit_list *Get the tests in the fit where the predictions fall outside of the error bars.*

Description

Get the tests in the fit where the predictions fall outside of the error bars.

Usage

```
## S3 method for class 'agraph_fit_list'
poor_fits(fit, sigma = 6)
```

Arguments

fit	The fitted graphs.
sigma	The width of the error bars.

Value

The poorly fitted tests.

print.agraph_fit *Print function for the fitted graph.*

Description

Prints the value of `cost_function` of the fitted graph, and complains if some or all of the admixture proportions aren't trurly fitted. Note: the admixture proportion not being trurly fitted does not necessarily mean that there is no evidence of an admix event!

Usage

```
## S3 method for class 'agraph_fit'
print(x, ...)
```

Arguments

x	The fitted object.
...	Additional parameters.

See Also

`link{summary.agraph_fit}`

`project_to_population` *Map sample names to population names.*

Description

Map sample names to population names.

Usage

```
project_to_population(data, f)
```

Arguments

`data`	The data frame to modify.
`f`	Function mapping sample names to population names.

Details

This function maps the sample names in `W`, `X`, `Y`, and `Z` to population names (typically what an admixture graph has for leaves) and stores the original sample names so we can map them back again after using the graph for making predictions.

`remove_duplicates` *Remove duplicate graphs from a list.*

Description

Using `canonise_graph` to calculate unique characteristic logical vector for each graph in a given list of graphs, then sorts the list according to this attribute and remove repeated graphs. Leaf names count so that graphs with permuted leaves are considered different. Also organises similar graphs next to each other if `organise` is `TRUE`, but this is extremely slow.

Usage

```
remove_duplicates(graph_list, organise = FALSE, return_piles = FALSE)
```

Arguments

`graph_list`	A list of graphs with the same leaf set.
`organise`	If `TRUE` also organises isomorphic graphs (now disregarding leaf names) next to each other.
`return_piles`	If `TRUE` and `organise` is also `TRUE`, the output will be a list of lists of isomorphic graphs instead of one big list.

Value

A list of graphs that are all different (or a list of lists of graphs if both `organise` and `return_piles` are `TRUE`).

`rename_nodes` *Rename nodes.*

Description

Changes the names of the nodes of a graph. Capable of giving new standard names to the inner nodes in a way that only depends on the graph topology (without the root) and the leaf names. This is necessary when detecting when graphs are identical up to inner node and admixture proportion names, see `canonise_graph` and `remove_duplicates`.

Usage

```
rename_nodes(graph, newnames = list())
```

Arguments

`graph`	The graph to be renamed.
`newnames`	A list of new names, given in the form `list(old = "new")`. Nodes not listed will keep their old name, unless no list is given at all, in which case the leaves keep their old names while the inner nodes get new standardised names.

Value

A graph with new node names.

`residuals.agraph_fit` *Errors of prediction in the fitted graph*

Description

Gets $f - F$, the difference between predicted and observed statistics, for each data point used in the fit.

Usage

```
## S3 method for class 'agraph_fit'
residuals(object, ...)
```

Arguments

`object`	The fitted object.
`...`	Additional parameters.

See Also

`link{summary.agraph_fit}`

`run_metropolis_hasting` *Run a Metropolis-Hasting MCMC to sample graph parameters.*

Description

The MCMC performs a random walk in transformed parameter space (edge lengths are log transformed and admixture proportions inverse Normal distribution transformed) and from this explores the posterior distribution of graph parameters.

Usage

```
run_metropolis_hasting(model, initial_state, iterations, no_temperatures = 1,
  cores = 1, no_flips = 1, max_tmp = 100, verbose = TRUE)
```

Arguments

`model`	Object constructed with `make_mcmc_model`.
`initial_state`	The initial set of graph parameters.
`iterations`	Number of iterations to sample.
`no_temperatures`	Number of chains in the MC3 procedure.
`cores`	Number of cores to spread the chains across. Best performance is when `cores = no_temperatures`.
`no_flips`	Mean number of times a flip between two chains should be proposed after each step.
`max_tmp`	The highest temperature.
`verbose`	Logical value determining if a progress bar should be shown during the run.

Details

Using the posterior distribution of parameters is one approach to getting parameter estimates and a sense of their variability. Credibility intervals can directly be obtained from sampled parameters; to get confidence intervals from the likelihood maximisation approach requires either estimating the Hessian matrix for the likelihood or a boot-strapping approach to the data.

From sampling the likelihood values for each sample from the posterior we can also compute the model likelihood (the probability of the data when integrating over all the model parameters). This gives us a direct way of comparing graphs since the ratio of likelihoods is the Bayes factor between the models. Comparing models using maximum likelihood estimtates is more problematic since usually graphs are not nested models.

Value

A matrix containing the trace of the chain with temperature 1.

seven_leaves_graphs *Seven leaves graphs.*

Description

Kind of obsolete since the introduction of all_graphs. A comprehensive listing of all the 48 admixture graphs with seven leaves and at most one admixture event. Our convention is that the position of the root does not matter (as long as it's not after an admixture event) and that graphs that have *eyes*, two inner nodes with the property that all the paths between any two leaves visits both or neither of them, are excluded. The reason is that the f statistics can't detect the exact position of the root or distinguish between an eye and a simple branch. The position of the root can be moved later with the function make_an_outgroup.

Usage

```
seven_leaves_graphs
```

Format

A list of functions on seven leaves and a parameter `permutations` which is `FALSE` by default. The outputs of these functions are either single agraph objects with the input vector as leaves, or if `permutations` is `TRUE`, lists of all the possible agraph objects with that leaf set up to symmetry.

See Also

all_graphs

make_permutations

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

add_an_admixture2

make_an_outgroup

Other graphs: eight_leaves_trees, five_leaves_graphs, four_leaves_graphs, six_leaves_graphs

Examples

```
# While the usage of this function is pretty self-explanatory, let's plot all the graphs
# just for browsing.
for (i in seq(1, length(seven_leaves_graphs))) {
  graph <- seven_leaves_graphs[[i]](c("A", "B", "C", "D", "E", "F", "G"))
  # This is how you include quotation marks in strings by the way:
  title <- paste("seven_leaves_graphs[[", i,
                 "]](c(\"A\", \"B\", \"C\", \"D\", \"E\", \"F\", \"G\"))", sep = "")
  plot(graph, color = "seagreen", title = title)
}
```

seven_leaves_trees *Seven leaves trees.*

Description

The function seven_leaves_graphs is better than this as it also contains graphs with one admixture. (This function is only kept for legacy reasons.) Even more obsolete since the introduction of all_graphs.

Usage

```
seven_leaves_trees
```

Format

A list of functions on seven leaves. The outputs of these functions are agraph objects.

sf2 *Calculate the f_2(A, B) statistics.*

Description

Calculate the $f_2(A, B)$ statistics.

Usage

```
sf2(graph, A, B)
```

Arguments

graph	The admixture graph.
A	A leaf node.
B	A leaf node.

Value

A symbolic representation of the equation for the f_2 statistics given by the admixture graph.

`sf3` *Calculate the f_3(A; B, C) statistics.*

Description

Calculate the $f_3(A; B, C)$ statistics.

Usage

```
sf3(graph, A, B, C)
```

Arguments

`graph`	The admixture graph.
`A`	A leaf node.
`B`	A leaf node.
`C`	A leaf node.

Value

A symbolic representation of the equation for the f_3 statistics given by the admixture graph.

`sf4` *Calculate the f_4(W, X; Y, Z) statistics.*

Description

Calculate the $f_4(W, X; Y, Z)$ statistics.

Usage

```
sf4(graph, W, X, Y, Z)
```

Arguments

`graph`	The admixture graph.
`W`	A leaf node.
`X`	A leaf node.
`Y`	A leaf node.
`Z`	A leaf node.

Value

A symbolic representation of the equation for the f_4 statistics given by the admixture graph.

six_leaves_graphs *Six leaves graphs.*

Description

Kind of obsolete since the introduction of all_graphs. A comprehensive listing of all the 21 admixture graphs with six leaves and at most one admixture event. Our convention is that the position of the root does not matter (as long as it's not after an admixture event) and that graphs that have *eyes*, two inner nodes with the property that all the paths between any two leaves visits both or neither of them, are excluded. The reason is that the f statistics can't detect the exact position of the root or distinguish between an eye and a simple branch. The position of the root can be moved later with the function make_an_outgroup.

Usage

```
six_leaves_graphs
```

Format

A list of functions on six leaves and a parameter `permutations` which is `FALSE` by default. The outputs of these functions are either single agraph objects with the input vector as leaves, or if `permutations` is `TRUE`, lists of all the possible agraph objects with that leaf set up to symmetry.

See Also

all_graphs

make_permutations

fit_permutations_and_graphs

fit_graph_list

add_a_leaf

add_an_admixture

add_an_admixture2

make_an_outgroup

Other graphs: eight_leaves_trees, five_leaves_graphs, four_leaves_graphs, seven_leaves_graphs

Examples

```
# While the usage of this function is pretty self-explanatory, let's plot all the graphs
# just for browsing.
for (i in seq(1, length(six_leaves_graphs))) {
  graph <- six_leaves_graphs[[i]](c("A", "B", "C", "D", "E", "F"))
  # This is how you include quotation marks in strings by the way:
  title <- paste("six_leaves_graphs[[", i,
                 "]](c(\"A\", \"B\", \"C\", \"D\", \"E\", \"F\"))", sep = "")
  plot(graph, color = "yellow4", title = title)
}
```

`split_population` *Reverse a projection of samples to populations.*

Description

Reverse a projection of samples to populations.

Usage

```
split_population(x)
```

Arguments

x	The projected data or a fitted object on projected data.

`split_population.agraph_fit` *Reverse a projection of samples to populations.*

Description

Reverse a projection of samples to populations.

Usage

```
## S3 method for class 'agraph_fit'
split_population(x)
```

Arguments

x	The projected data or a fitted object on projected data.

split_population.data.frame

Reverse a projection of samples to populations.

Description

Reverse a projection of samples to populations.

Usage

```
## S3 method for class 'data.frame'
split_population(x)
```

Arguments

x	The projected data or a fitted object on projected data.

summary.agraph_fit

Summary for the fitted graph.

Description

Prints:
Optimal admixture proportions and a complaint if some of them are not trurly fitted, *i. e.* if after fixing a (possibly empty) subset of them, the rest have typically no effect on the cost function. Here typically means that some isolated values of the admixture proportions, like 0 or 1, might actually give a significantly worse fit than the constant fit given by any other values (but not better). Thus, an admixture proportion not being fitted does not always mean that there is no evidence of an admix event, as fixing them at 0 or 1 could make the fit worse while the exact value won't matter otherwise.
The optimal edge lengths give one of the solutions for the best fit. It is generally not unique, as after fixing the admixture proportions, the best edge lengths are a non-negative least square solution for a system of linear equations. To get all the solutions one has to add any solution of the corresponding homogeneous system to the given exaple solution (and exclude possible negative values). The solutions of the homogeneous system are given as a set of free edge lengths that may obtain any non-negative value, and bounded edge lengths that linearly depend on the free ones.
Minimal error is the value of the `cost_function` at the fit.

Usage

```
## S3 method for class 'agraph_fit'
summary(object, ...)
```

Arguments

object	The fitted object.
...	Additional parameters.

`sum_of_squared_errors` *Get the sum of squared errors for a fitted graph.*

Description

Get the sum of squared errors for a fitted graph.

Usage

```
sum_of_squared_errors(x)
```

Arguments

`x`	The fitted graph.

Value

SSE for the fit.

`sum_of_squared_errors.agraph_fit` *Get the sum of squared errors for a fitted graph.*

Description

Get the sum of squared errors for a fitted graph.

Usage

```
## S3 method for class 'agraph_fit'
sum_of_squared_errors(x)
```

Arguments

`x`	The fitted graph.

Value

SSE for the fit.

`sum_of_squared_errors.agraph_fit_list` *Get the sum of squared errors for a list of fitted graph.*

Description

Get the sum of squared errors for a list of fitted graph.

Usage

```
## S3 method for class 'agraph_fit_list'
sum_of_squared_errors(x)
```

Arguments

`x`	The fitted graphs.

Value

SSE for the fits.

`thinning` *Thins out an MCMC trace.*

Description

Thins out an MCMC trace.

Usage

```
thinning(trace, k)
```

Arguments

`trace`	A trace from an MCMC run.
`k`	The number of lines to skip over per retained sample.

`vector_to_graph` *Vector to graph.*

Description

Interprets a logical vector back to an `agraph` object. The admixture proportion names are now lost.

Usage

```
vector_to_graph(vector)
```

Arguments

`vector`	A logical vector.

Value

The graph corresponding to the vector.

Index

∗Topic **datasets**
bears, 16
eight_leaves_trees, 22
five_leaves_graphs, 36
four_leaves_graphs, 38
graphs_2_0, 39
graphs_3_0, 40
graphs_3_1, 40
graphs_4_0, 41
graphs_4_1, 41
graphs_4_2, 42
graphs_5_0, 42
graphs_5_1, 43
graphs_5_2, 43
graphs_6_0, 44
graphs_6_1, 44
graphs_6_2, 45
graphs_7_0, 45
graphs_7_1, 46
graphs_8_0, 46
seven_leaves_graphs, 72
seven_leaves_trees, 73
six_leaves_graphs, 75

add_a_leaf, *5*, *8*, 8, *22*, *37*, *38*, *52*, *53*, *72*, *75*
add_an_admixture, 5, *7*, *8*, *22*, *37*, *38*, *52*, *53*, *72*, *75*
add_an_admixture2, *5*, 7, *8*, *22*, *37*, *38*, *52*, *53*, *72*, *75*
add_graph_f4, 9
add_graph_f4_sign, 10
admix_props, 11, *12*
admixture_edge, 10, *12*
admixture_proportions, 11, *12*
admixturegraph-package, 4
agraph, 12, *22*, *28*, *29*, *32*, *33*, *36*, *38–47*, *63*, *72*, *73*, *75*, *80*
agraph_children, 13, *14*
agraph_parents, *13*, 13
agraph_weights, 14

all_graphs, *5*, *8*, 14, *22*, *34*, *36*, *38–46*, *72*, *73*, *75*
all_path_overlaps, 15, *48*, *49*, *60*
all_paths, 15

bears, 16
build_edge_optimisation_matrix, 16, *20*, *21*, *24*, *33*, *50*
burn_in, 17

calculate_concentration, 17, *29*, *33*, *50*
canonise_expression, 18
canonise_graph, 18, *69*, *70*
character, *18*
coef.agraph_fit, 19
contour, *66*
cost_function, 19, *21*, *28*, *29*, *32*, *33*, *50*, *65*, *66*, *68*, *77*

edge, *12*, 20
edge_optimisation_function, *19*, *20*, 21, *50*
eight_leaves_trees, 22, *35*, *37*, *38*, *52*, *53*, *72*, *75*
eval, *16*
evaluate_f4, 23
examine_edge_optimisation_matrix, 23, *32*
extract_admixture_proportion_parameters, 24
extract_graph_parameters, *16*, 25, *47*
extract_trees, 25

f2, 26
f3, 26
f4, *18*, 27
f4stats, 27
fast_fit, 28, *32–35*
fast_plot, 30, *62*, *63*
filter_on_leaves, 31

fit_graph, *28*, *29*, 32
fit_graph_list, *5*, *8*, *15*, *22*, 34, *35*, *36*, *38*, *52*, *72*, *75*
fit_permutations_and_graphs, *5*, *8*, *22*, *34*, 35, *36*, *38*, *52*, *53*, *72*, *75*
fitted.agraph_fit, 31
five_leaves_graphs, *22*, *35*, 36, *38*, *52*, *53*, *72*, *75*
format_path, 37
four_leaves_graphs, *22*, *35*, *37*, 38, *52*, *53*, *72*, *75*

get_graph_f4_sign, 39
ginv, *56*
graph_environment, 47
graph_to_vector, *39–46*, 47
graphs_2_0, 39
graphs_3_0, 40
graphs_3_1, 40
graphs_4_0, 41
graphs_4_1, 41
graphs_4_2, 42
graphs_5_0, 42
graphs_5_1, 43
graphs_5_2, 43
graphs_6_0, 44
graphs_6_1, 44
graphs_6_2, 45
graphs_7_0, 45
graphs_7_1, 46
graphs_8_0, 46

is_descendant_of, 48
is_negative, 48
is_positive, 49
is_unknown, 49
is_zero, 49

log_likelihood, *20*, *21*, 50
log_sum_of_logs, 51
lsqnonneg, *56*

make_an_outgroup, *5*, *8*, *22*, *36–38*, 51, *72*, *75*
make_mcmc_model, 52, *71*
make_permutations, *22*, *35*, *36*, *38*, *52*, 53, *72*, *75*
model_bayes_factor_n, 54
model_likelihood, 55
model_likelihood_n, 55

mynonneg, *20*, *21*, *28*, *32*, 56

no_admixture_events, 57
no_admixture_events.agraph, 57
no_admixture_events.agraph_fit, 58
no_admixture_events.agraph_fit_list, 58
no_poor_fits, 59
no_poor_fits.agraph_fit, 59
no_poor_fits.agraph_fit_list, 60

optimset, *29*, *33*
overlaps_sign, 60

parent_edges, *12*, 61
parse, *16*
path_overlap, 61
plot.agraph, *12*, *30*, 62
plot.agraph_fit, 64
plot.f4stats, 65
plot_fit_1, 65, *66*
plot_fit_2, *66*, 66
poor_fits, 67
poor_fits.agraph_fit, 67
poor_fits.agraph_fit_list, 68
print.agraph_fit, 68
project_to_population, 69

qr.solve, *24*, *56*

remove_duplicates, 69, *70*
rename_nodes, 70
residuals.agraph_fit, 70
run_metropolis_hasting, 71

seven_leaves_graphs, *22*, *35*, *37*, *38*, *52*, 72, *73*, *75*
seven_leaves_trees, *53*, 73
sf2, 73
sf3, 74
sf4, 74
six_leaves_graphs, *22*, *35*, *37*, *38*, *52*, *53*, *72*, 75
split_population, 76
split_population.agraph_fit, 76
split_population.data.frame, 77
sum_of_squared_errors, 78
sum_of_squared_errors.agraph_fit, 78
sum_of_squared_errors.agraph_fit_list, 79

summary.agraph_fit, *24*, *33*, 77

thinning, 79

vector_to_graph, *39–46*, 80