Package 'BrainNetTest'

April 23, 2026

Type Package

Title Hypothesis Testing for Populations of Brain Networks

Version 0.2.0

Description Non-parametric hypothesis testing for populations of brain networks represented as graphs, following the L1-distance ANOVA framework of Fraiman and Fraiman (2018) <doi:10.1038/s41598-018-21688-0>. The package builds on this nonparametric graph-comparison framework, extending it with procedures for edge-level inference and identification of the specific connections driving group differences. In particular, it provides utilities to compute central (mean) graphs, pairwise Manhattan distances between adjacency matrices, the group test statistic T, and a fast permutation procedure to identify the critical edges that drive between-group differences. Helper functions to generate synthetic community-structured graphs and to visualise brain networks with communities are also included.

License MIT + file LICENSE

URL https://github.com/mmaximiliano/BrainNetTest

BugReports https://github.com/mmaximiliano/BrainNetTest/issues

Encoding UTF-8

Depends R (>= 4.0.0)

Imports stats, graphics, igraph

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Maximiliano Martino [aut, cre] (ORCID: <https://orcid.org/0000-0002-2437-4387>), Daniel Fraiman [aut] (ORCID: <https://orcid.org/0000-0002-0482-9137>)

Maintainer Maximiliano Martino <maxii.martino@gmail.com>

Repository CRAN

Date/Publication 2026-04-23 16:30:02 UTC

Contents

`compute_central_graph` *Compute the Central (Representative) Graph for a Population*

Description

This function computes the central graph for a given population by averaging the adjacency matrices of all graphs within that population.

Usage

```
compute_central_graph(graph_list)
```

Arguments

`graph_list`	A list of adjacency matrices representing brain networks.

Value

A single adjacency matrix representing the central graph of the population.

Examples

```
# Generate synthetic Control data
Control <- list(
  generate_random_graph(n_nodes = 5, edge_prob = 0.1),
  generate_random_graph(n_nodes = 5, edge_prob = 0.1)
)
central_graph <- compute_central_graph(Control)
```

compute_distance *Compute the Manhattan Norm Distance Between Two Graphs*

Description

This function calculates the Manhattan norm (L1 norm) distance between two adjacency matrices representing brain networks.

Usage

```
compute_distance(G, M)
```

Arguments

G	A square adjacency matrix representing a brain network.
M	A square adjacency matrix representing the central brain network.

Value

A numeric value representing the Manhattan distance between G and M.

Examples

```
# Generate synthetic data
G1 <- generate_random_graph(n_nodes = 5, edge_prob = 0.1)
G2 <- generate_random_graph(n_nodes = 5, edge_prob = 0.1)
central_graph <- compute_central_graph(list(G1, G2))
distance <- compute_distance(G1, central_graph)
```

compute_edge_frequencies *Compute Edge Frequencies in Populations*

Description

This function computes the frequency of each edge (connection between nodes) across all graphs within each population.

Usage

```
compute_edge_frequencies(populations)
```

Arguments

populations	A list where each element is a population containing a list of graphs (adjacency matrices).

Value

A list containing edge counts and edge proportions for each population.

Examples

```
# Generate synthetic populations
control_graphs <- generate_category_graphs(
  n_graphs = 5,
  n_nodes = 10,
  n_communities = 2,
  base_intra_prob = 0.8,
  base_inter_prob = 0.2,
  intra_prob_variation = 0.05,
  inter_prob_variation = 0.05,
  seed = 1
)
patient_graphs <- generate_category_graphs(
  n_graphs = 5,
  n_nodes = 10,
  n_communities = 2,
  base_intra_prob = 0.6,
  base_inter_prob = 0.4,
  intra_prob_variation = 0.05,
  inter_prob_variation = 0.05,
  seed = 2
)
populations <- list(Control = control_graphs, Patient = patient_graphs)

# Compute edge frequencies
frequencies <- compute_edge_frequencies(populations)
# View edge counts for the first population
edge_counts_control <- frequencies$edge_counts[,,1]
print(edge_counts_control)
```

---

compute_edge_pvalues *Compute P-Values for Edge Differences Between Populations*

Description

This function computes p-values for the differences in edge presence between populations using statistical tests (e.g., Fisher's exact test, chi-squared test).

Usage

```
compute_edge_pvalues(edge_counts, N, method = "fisher", adjust_method = "none")
```

Arguments

edge_counts	An array of edge counts from compute_edge_frequencies.
N	A vector of sample sizes for each population.
method	The statistical test method to use: "fisher", "chi.squared", or "prop". Default is "fisher".
adjust_method	The method for p-value adjustment for multiple testing. Default is "none".

Value

A matrix of p-values for each edge.

Examples

```
# Generate synthetic populations
control_graphs <- generate_category_graphs(
  n_graphs = 5,
  n_nodes = 10,
  n_communities = 2,
  base_intra_prob = 0.8,
  base_inter_prob = 0.2,
  intra_prob_variation = 0.05,
  inter_prob_variation = 0.05,
  seed = 1
)
patient_graphs <- generate_category_graphs(
  n_graphs = 5,
  n_nodes = 10,
  n_communities = 2,
  base_intra_prob = 0.6,
  base_inter_prob = 0.4,
  intra_prob_variation = 0.05,
  inter_prob_variation = 0.05,
  seed = 2
)
populations <- list(Control = control_graphs, Patient = patient_graphs)

# Compute edge frequencies
frequencies <- compute_edge_frequencies(populations)
edge_counts <- frequencies$edge_counts
N <- sapply(populations, length)

# Compute p-values for edge differences
edge_pvalues <- compute_edge_pvalues(edge_counts, N)
# View p-values for the first few edges
print(edge_pvalues[1:5, 1:5])
```

`compute_test_statistic` *Compute the Test Statistic T for Brain Network Populations*

Description

This function computes the test statistic T to assess whether different populations of brain networks originate from the same distribution.

Usage

```
compute_test_statistic(populations, a = 1)
```

Arguments

`populations`	A named list where each element is a list of adjacency matrices for a population. Example: `list(Control = list(G1, G2, ...), PatientA = list(G1, G2, ...), PatientB = list(G1, G2, ...))`
`a`	A normalization constant. Default is 1.

Value

A numeric value representing the test statistic T.

Examples

```
# Generate synthetic populations data
Control <- list(
  generate_random_graph(n_nodes = 5, edge_prob = 0.1),
  generate_random_graph(n_nodes = 5, edge_prob = 0.1)
)
PatientA <- list(
  generate_random_graph(n_nodes = 5, edge_prob = 0.15),
  generate_random_graph(n_nodes = 5, edge_prob = 0.15)
)
PatientB <- list(
  generate_random_graph(n_nodes = 5, edge_prob = 0.2),
  generate_random_graph(n_nodes = 5, edge_prob = 0.2)
)

populations <- list(Control = Control, PatientA = PatientA, PatientB = PatientB)

# Compute the test statistic T
T_value <- compute_test_statistic(populations, a = 1)
print(T_value)
```

generate_category_graphs *Generate a Set of Graphs with Similar Community Structures for a Category*

Description

This function generates a list of adjacency matrices representing brain networks belonging to the same category (e.g., Control group). The generated graphs have similar community structures but vary slightly in their intra-community and inter-community connection probabilities to reflect natural variability.

Usage

```
generate_category_graphs(
  n_graphs = 10,
  n_nodes = 100,
  n_communities = 4,
  community_sizes = NULL,
  base_intra_prob = 0.8,
  base_inter_prob = 0.2,
  intra_prob_variation = 0.05,
  inter_prob_variation = 0.05,
  seed = NULL
)
```

Arguments

n_graphs	An integer specifying the number of graphs to generate. Default is 10.
n_nodes	An integer specifying the total number of nodes (brain regions). Default is 100.
n_communities	An integer specifying the number of communities. Default is 4.
community_sizes	An integer vector specifying the sizes of each community. If NULL, communities are of equal size. Default is NULL.
base_intra_prob	A numeric value between 0 and 1, or a numeric vector of length n_communities, specifying the base probability of an edge existing between nodes within the same community. If a vector, each element sets the base probability for the corresponding community. Default is 0.8.
base_inter_prob	A numeric value between 0 and 1 specifying the base probability of an edge existing between nodes from different communities. Default is 0.2.
intra_prob_variation	A numeric value specifying the maximum variation to apply to the intra-community probability for each graph. Default is 0.05.

`inter_prob_variation`
: A numeric value specifying the maximum variation to apply to the inter-community probability for each graph. Default is 0.05.

`seed`
: An optional integer for setting the random seed to ensure reproducibility. Default is NULL.

Value

A list of symmetric binary adjacency matrices with no self-loops, representing brain networks with similar community structures.

Examples

```
# Generate a set of 5 graphs for the Control category
control_graphs <- generate_category_graphs(n_graphs = 5, n_nodes = 100, n_communities = 4,
                                           base_intra_prob = 0.8, base_inter_prob = 0.2)
```

---

generate_community_graph

Generate a Random Symmetric Adjacency Matrix with Community Structure

---

Description

This function generates a random symmetric adjacency matrix representing a brain network with community structure. Nodes within the same community have a higher probability of being connected compared to nodes from different communities.

Usage

```
generate_community_graph(
  n_nodes = 100,
  n_communities = 4,
  community_sizes = NULL,
  intra_prob = 0.8,
  inter_prob = 0.2,
  seed = NULL
)
```

Arguments

`n_nodes`
: An integer specifying the total number of nodes (brain regions). Default is 100.

`n_communities`
: An integer specifying the number of communities. Default is 4.

`community_sizes`
: An integer vector specifying the sizes of each community. If NULL, communities are of equal size. Default is NULL.

intra_prob	A numeric value between 0 and 1, or a numeric vector of length n_communities, specifying the probability of an edge existing between nodes within the same community. If a scalar, the same probability is used for all communities. Default is 0.8.
inter_prob	A numeric value between 0 and 1 specifying the probability of an edge existing between nodes from different communities. Default is 0.2.
seed	An optional integer for setting the random seed to ensure reproducibility. Default is NULL.

Value

A symmetric binary adjacency matrix with no self-loops, representing a brain network with community structure.

Examples

```
# Generate a brain network with community structure
G <- generate_community_graph(n_nodes = 100, n_communities = 4, intra_prob = 0.8, inter_prob = 0.2)
```

---

generate_random_graph *Generate a Random Symmetric Adjacency Matrix*

Description

This function generates a random symmetric adjacency matrix representing a brain network. The adjacency matrix is binary, with edges present based on a specified probability.

Usage

```
generate_random_graph(n_nodes, edge_prob = 0.1)
```

Arguments

n_nodes	An integer specifying the number of nodes (brain regions).
edge_prob	A numeric value between 0 and 1 specifying the probability of an edge existing between any two nodes.

Value

A symmetric binary adjacency matrix with no self-loops.

Examples

```
G <- generate_random_graph(n_nodes = 10, edge_prob = 0.1)
```

`get_critical_nodes` *Extract Critical Nodes from Critical Edge Results*

Description

Given the output of `identify_critical_links`, this function identifies the unique nodes involved in the critical edges and summarizes their participation. Each node is reported with the number of critical edges it participates in (its *critical degree*). If node labels are supplied, they are included in the output.

Usage

```
get_critical_nodes(result, node_labels = NULL)
```

Arguments

`result`	The list returned by `identify_critical_links`. Must contain a `critical_edges` component (a data frame with columns `node1` and `node2`, or `NULL`).
`node_labels`	An optional character vector of length p (the number of nodes in the network), where `node_labels[i]` is the label for node `i`. If `NULL` (the default), no labels are included.

Value

A data frame with the following columns, ordered by decreasing `critical_degree`:

node Integer node index.

label Character label for the node (only present when `node_labels` is not `NULL`).

critical_degree Number of critical edges incident on this node.

If no critical edges were found (`result$critical_edges` is `NULL` or has zero rows), an empty data frame with the same columns is returned.

Examples

```
# Generate two synthetic populations with different community structure
control <- generate_category_graphs(n_graphs = 15, n_nodes = 10,
  n_communities = 2, base_intra_prob = 0.8, base_inter_prob = 0.2, seed = 1)
patient <- generate_category_graphs(n_graphs = 15, n_nodes = 10,
  n_communities = 2, base_intra_prob = 0.5, base_inter_prob = 0.5, seed = 2)
populations <- list(Control = control, Patient = patient)

# Identify critical edges
result <- identify_critical_links(populations, alpha = 0.05,
  method = "fisher", n_permutations = 200, seed = 42)

# Extract critical nodes (no labels)
get_critical_nodes(result)
```

```
# With labels
labels <- paste0("Region_", 1:10)
get_critical_nodes(result, node_labels = labels)
```

`identify_critical_links` *Identify Critical Edges That Explain Population Differences*

Description

Iteratively identifies the edges that drive the observed difference between two or more populations of brain networks, using the permutation-based L1-distance ANOVA test of Fraiman and Fraiman (2018). Edges are ranked by their marginal p-value, then removed in batches until the global test is no longer significant. The permutation null is re-evaluated incrementally via a prefix-sum decomposition of T, which avoids recomputing the statistic from scratch at every step.

Usage

```
identify_critical_links(
  populations,
  alpha = 0.05,
  method = "fisher",
  adjust_method = "none",
  batch_size = 1,
  n_permutations = 1000,
  a = 1,
  seed = NULL
)
```

Arguments

`populations`	A named `list` with at least two elements. Each element is itself a list of square, binary, symmetric adjacency matrices (zero diagonal, no self-loops). All graphs must share the same number of nodes.
`alpha`	Numeric significance level used both for the global test and for the iterative edge-removal stopping rule. Default `0.05`.
`method`	Character string selecting the marginal edge test used to rank candidate edges. One of `"fisher"` (default), `"chi.squared"`, or `"prop"`. See `compute_edge_pvalues`.
`adjust_method`	Character string passed to `p.adjust` for multiplicity correction of the marginal p-values. Default `"none"`.
`batch_size`	Positive integer giving the number of edges to drop per iteration. Larger values speed up the procedure at the cost of coarser resolution. Default `1`.
`n_permutations`	Positive integer giving the number of permutation replicates used to approximate the null distribution of T. Default `1000`.

a	Positive numeric normalisation constant for the test statistic T (see compute_test_statistic). Default 1.
seed	Optional integer seed for reproducibility of the permutation replicates. Default NULL.

Details

The implementation exploits the fact that T decomposes as a sum of per-edge contributions Δ_e. Removing an edge e is equivalent to subtracting Δ_e from T, so prefix sums of the sorted Δ_e values give the test statistic after any number of removals in O(1). The permutation null is built once via a single BLAS matrix multiplication and reused for every iteration, reducing the overall complexity from O(K * B * |E| * m) (naive) to O(B * |E| * m).

Permutation replicates are generated by uniformly permuting the group labels across the pooled sample of all graphs, while preserving the original group sizes (sampling without replacement).

Value

A list with three components: critical_edges (a data.frame with columns node1, node2, p_value, listing the edges removed until the test became non-significant, ordered from most to least significant; NULL if the global test was not significant), edges_removed (a list of length-2 integer vectors giving the (i, j) indices of the removed edges, in removal order), and modified_populations (the input populations with the critical edges zeroed out in every graph).

References

Fraiman, D. and Fraiman, R. (2018) An ANOVA approach for statistical comparisons of brain networks. *Scientific Reports*, 8, 4746. doi:10.1038/s41598018216880.

See Also

compute_test_statistic, compute_edge_pvalues, get_critical_nodes.

Examples

```
set.seed(1)
control <- generate_category_graphs(
  n_graphs = 15, n_nodes = 10, n_communities = 2,
  base_intra_prob = 0.8, base_inter_prob = 0.2, seed = 1)
patient <- generate_category_graphs(
  n_graphs = 15, n_nodes = 10, n_communities = 2,
  base_intra_prob = 0.5, base_inter_prob = 0.5, seed = 2)
populations <- list(Control = control, Patient = patient)
result <- identify_critical_links(
  populations, alpha = 0.05, method = "fisher",
  n_permutations = 200, seed = 42)
head(result$critical_edges)
```

plot_critical_edges *Visualize Central Graphs and Critical Edges*

Description

Produces a multi-panel figure summarising the output of identify_critical_links: one panel per population showing the weighted central graph, plus a final panel that highlights the critical edges on a chosen reference central graph. This is the canonical visualization of the **BrainNetTest** workflow and replaces the manual **igraph** plumbing required to build the same figure.

Usage

```
plot_critical_edges(
  populations,
  critical_links,
  communities = NULL,
  layout = NULL,
  central_graphs = NULL,
  reference = 1L,
  threshold = 0.5,
  edge_scale = 6,
  critical_color = "red",
  critical_width = 3,
  background_color = "grey80",
  background_width = 1,
  vertex_size = 12,
  vertex_label = TRUE,
  panel_titles = NULL,
  mfrow = NULL,
  ...
)
```

Arguments

populations	A named list of populations of adjacency matrices, as accepted by identify_critical_links. Used to compute the per-population central graphs unless central_graphs is supplied.
critical_links	The object returned by identify_critical_links. Its critical_edges component provides the edges to highlight in the final panel.
communities	Optional integer (or factor) vector of length n_nodes giving the community membership of every node. Used to color vertices consistently across panels. Default NULL (all vertices share the default **igraph** color).
layout	Optional layout for the graphs. Either an n_nodes x 2 numeric matrix of vertex coordinates (recommended, so that all panels share the same node positions) or an **igraph** layout function such as layout_with_fr. Default layout_in_circle.

`central_graphs` Optional list of pre-computed central graphs (one per population). If `NULL`, they are computed via `compute_central_graph`.

`reference` Either an integer index or a name selecting which population's central graph is used as the background of the critical-edges panel. Default `1L` (the first population).

`threshold` Numeric in `[0, 1]`. The reference central graph is binarised by `> threshold` for the critical-edges panel. Default `0.5`.

`edge_scale` Numeric multiplier applied to the central-graph edge weights to set `edge.width` in the per-population panels. Default `6`.

`critical_color`, `critical_width`
Color and line width used to draw critical edges. Defaults `"red"` and `3`.

`background_color`, `background_width`
Color and line width used to draw non-critical edges in the critical panel. Defaults `"grey80"` and `1`.

`vertex_size` Vertex size passed to `plot.igraph`. Default `12`.

`vertex_label` Logical. If `TRUE` (default) vertex indices are shown; otherwise vertex labels are suppressed.

`panel_titles` Optional character vector of length `length(populations) + 1` with custom panel titles. Default uses `paste(names(populations), "Central Graph")` followed by `"Critical edges"`.

`mfrow` Optional integer vector of length 2 giving the `c(nrow, ncol)` arrangement passed to `par`. By default a near-square grid is chosen automatically (e.g.\ a `2 x 2` grid for two populations, matching the standard Control / Patient / Critical layout).

`...` Additional arguments forwarded to `plot.igraph`. Note that `edge.width`, `vertex.color`, `vertex.size`, `vertex.label` and `layout` are managed by this function and should not be passed via `....`

Details

The first `length(populations)` panels show the weighted central graph of each population, with edge widths proportional to the central-graph weights. The final panel binarises the chosen reference central graph at `threshold` and overlays the critical edges returned by `identify_critical_links()` in `critical_color`. Critical edges that are absent from the binarised reference graph are added so that they remain visible.

The graphics state (`par(mfrow, mar, oma)`) is restored on exit.

Value

Invisibly returns `NULL`; called for its side effect of producing the multi-panel plot.

See Also

`identify_critical_links`, `compute_central_graph`, `get_critical_nodes`.

Examples

```
set.seed(123)
community_sizes <- c(4, 2, 3, 3, 5)
control <- generate_category_graphs(
  n_graphs = 50, n_nodes = 17, n_communities = 5,
  community_sizes = community_sizes,
  base_intra_prob = rep(0.70, 5), base_inter_prob = 0.05,
  intra_prob_variation = 0.02, inter_prob_variation = 0.01)
patient <- generate_category_graphs(
  n_graphs = 50, n_nodes = 17, n_communities = 5,
  community_sizes = community_sizes,
  base_intra_prob = c(0.40, 0.70, 0.70, 0.70, 0.70),
  base_inter_prob = 0.05,
  intra_prob_variation = 0.02, inter_prob_variation = 0.01)
populations <- list(Control = control, Patient = patient)
result <- identify_critical_links(populations, alpha = 0.05,
  method = "fisher", n_permutations = 200, seed = 1)
communities <- rep(seq_along(community_sizes), times = community_sizes)
plot_critical_edges(populations, result, communities = communities)
```

---

rank_edges *Rank Edges Based on P-Values*

Description

This function ranks edges based on their p-values obtained from statistical tests, ordering them from lowest to highest p-value (most to least significant).

Usage

```
rank_edges(edge_pvalues)
```

Arguments

edge_pvalues	A square matrix of p-values for each edge, typically obtained from compute_edge_pvalues.

Value

A data frame with edges and their corresponding p-values, ordered from most significant.

Examples

```
# Generate synthetic populations
control_graphs <- generate_category_graphs(
  n_graphs = 5,
  n_nodes = 10,
  n_communities = 2,
  base_intra_prob = 0.8,
```

```
  base_inter_prob = 0.2,
  seed = 1
)
patient_graphs <- generate_category_graphs(
  n_graphs = 5,
  n_nodes = 10,
  n_communities = 2,
  base_intra_prob = 0.6,
  base_inter_prob = 0.4,
  seed = 2
)
populations <- list(Control = control_graphs, Patient = patient_graphs)

# Compute edge frequencies
frequencies <- compute_edge_frequencies(populations)
edge_counts <- frequencies$edge_counts
N <- sapply(populations, length)

# Compute p-values for edge differences
edge_pvalues <- compute_edge_pvalues(edge_counts, N)

# Rank edges based on p-values
edge_df <- rank_edges(edge_pvalues)
# View the top ranked edges
head(edge_df)
```

Index