Package: nevada (via r-universe)

January 1, 2025

Type Package

Title Network-Valued Data Analysis

Version 0.2.0.9000

Description A flexible statistical framework for network-valued data analysis. It leverages the complexity of the space of distributions on graphs by using the permutation framework for inference as implemented in the 'flipr' package. Currently, only the two-sample testing problem is covered and generalization to k samples and regression will be added in the future as well. It is a 4-step procedure where the user chooses a suitable representation of the networks, a suitable metric to embed the representation into a metric space, one or more test statistics to target specific aspects of the distributions to be compared and a formula to compute the permutation p-value. Two types of inference are provided: a global test answering whether there is a difference between the distributions that generated the two samples and a local test for localizing differences on the network structure. The latter is assumed to be shared by all networks of both samples. References: Lovato, I., Pini, A., Stamm, A., Vantini, S. (2020) ``Model-free two-sample test for network-valued data'' <doi:10.1016/j.csda.2019.106896>; Lovato, I., Pini, A., Stamm, A., Taquet, M., Vantini, S. (2021) ``Multiscale null hypothesis testing for network-valued data: Analysis of brain networks of patients with autism'' <doi:10.1111/rssc.12463>.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports igraph, Rcpp, tidyr, dplyr, purrr, tibble, forcats, ggplot2, rlang, flipr, cli, withr, tsne, umap, furrr, rgeomstats, tidygraph

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

URL https://astamm.github.io/nevada/,
https://github.com/astamm/nevada/

BugReports https://github.com/astamm/nevada/issues/

Suggests knitr, rmarkdown, covr, quarto

VignetteBuilder quarto

LinkingTo Rcpp, RcppArmadillo

Config/pak/sysreqs git libglpk-dev make libgit2-dev libicu-dev libpng-dev libxml2-dev libssl-dev python3 libx11-dev

Repository https://permaverse.r-universe.dev

RemoteUrl https://github.com/permaverse/nevada

RemoteRef HEAD

RemoteSha 109dbc64f2903d41b1159e8b4b1d4d271a9a633c

Contents

`as_nvd` *Coercion to Network-Valued Data Object*

Description

This function flags a list of `igraph` objects as an `nvd` object as defined in this package.

Usage

```
as_nvd(obj)
```

Arguments

`obj`	A list of `igraph` objects.

Value

An `nvd` object.

Examples

```
params <- list(n_dim = 1L, dim_size = 4L, order = 4L, p_rewire = 0.15)
out <- nvd(sample_size = 1L, model = "smallworld", !!!params)
as_nvd(out)
```

`as_vertex_partition` *Coercion to Vertex Partition*

Description

This function converts a vector of memberships into a proper vertex partition object.

Usage

```
as_vertex_partition(x)
```

Arguments

`x`	A list grouping the vertices by partition element or an integer or character vector of vertex memberships.

Value

A `vertex_partition` object storing the corresponding vertex partition.

Examples

```
m1 <- c("P1", "P3", "P4", "P1", "P2", "P2", "P3", "P1", "P4", "P3")
V1 <- as_vertex_partition(m1)
m2 <- as.integer(c(1, 3, 4, 1, 2, 2, 3, 1, 4, 3))
V2 <- as_vertex_partition(m2)
```

---

distances *Distances Between Networks*

Description

This is a collection of functions computing the distance between two networks.

Usage

```
dist_hamming(x, y, representation = "laplacian")

dist_frobenius(
  x,
  y,
  representation = "laplacian",
  matching_iterations = 0,
  target_matrix = NULL
)

dist_spectral(x, y, representation = "laplacian")

dist_root_euclidean(x, y, representation = "laplacian")
```

Arguments

`x`	A `tidygraph::tbl_graph` object or a matrix representing an underlying network.
`y`	A `tidygraph::tbl_graph` object or a matrix representing an underlying network. Should have the same number of vertices as `x`.
`representation`	A string specifying the desired type of representation, among: `"adjacency"`, `"laplacian"`, `"modularity"` or `"graphon"`. Default is `"laplacian"`.
`matching_iterations`	An integer value specifying the maximum number of runs when looking for the optimal permutation for graph matching. Defaults to `0L` in which case no matching is done.
`target_matrix`	A square numeric matrix of size `n` equal to the order of the graphs specifying a target matrix towards which the initial doubly stochastic matrix is shrunk each time the graph matching algorithm fails to provide a good minimum. Defaults to `NULL` in which case the target matrix is automatically chosen between the identity matrix or the uniform matrix on the n-simplex.

Details

Let X be the matrix representation of network x and Y be the matrix representation of network y. The Hamming distance between x and y is given by

$$\frac{1}{N(N-1)}\sum_{i,j}|X_{ij}-Y_{ij}|,$$

where N is the number of vertices in networks x and y. The Frobenius distance between x and y is given by

$$\sqrt{\sum_{i,j}(X_{ij}-Y_{ij})^2}.$$

The spectral distance between x and y is given by

$$\sqrt{\sum_{i}(a_i-b_i)^2},$$

where a and b of the eigenvalues of X and Y, respectively. This distance gives rise to classes of equivalence. Consider the spectral decomposition of X and Y:

$$X = VAV^{-1}$$

and

$$Y = UBU^{-1},$$

where V and U are the matrices whose columns are the eigenvectors of X and Y, respectively and A and B are the diagonal matrices with elements the eigenvalues of X and Y, respectively. The root-Euclidean distance between x and y is given by

$$\sqrt{\sum_{i}(V\sqrt{A}V^{-1}-U\sqrt{B}U^{-1})^2}.$$

Root-Euclidean distance can used only with the laplacian matrix representation.

Value

A scalar measuring the distance between the two input networks.

Examples

```
g1 <- igraph::sample_gnp(20, 0.1)
g2 <- igraph::sample_gnp(20, 0.2)
dist_hamming(g1, g2, "adjacency")
dist_frobenius(g1, g2, "adjacency")
dist_spectral(g1, g2, "laplacian")
dist_root_euclidean(g1, g2, "laplacian")
```

`dist_nvd` *Pairwise Distance Matrix Between Two Samples of Networks*

Description

This function computes the matrix of pairwise distances between all the elements of the two samples put together. The cardinality of the fist sample is denoted by n_1 and that of the second one is denoted by n_2.

Usage

```
dist_nvd(
  x,
  y = NULL,
  representation = "adjacency",
  distance = "frobenius",
  matching_iterations = 0,
  target_matrix = NULL
)
```

Arguments

`x`	A `base::list` of `tidygraph::tbl_graph` objects or matrix representations of underlying networks from a given first population.
`y`	A `base::list` of `tidygraph::tbl_graph` objects or matrix representations of underlying networks from a given second population.
`representation`	A string specifying the desired type of representation, among: `"adjacency"`, `"laplacian"`, `"modularity"` or `"graphon"`. Default is `"laplacian"`.
`distance`	A string specifying the chosen distance for calculating the test statistic, among: `"hamming"`, `"frobenius"`, `"spectral"` and `"root-euclidean"`. Default is `"frobenius"`.
`matching_iterations`	An integer value specifying the maximum number of runs when looking for the optimal permutation for graph matching. Defaults to `0L` in which case no matching is done.
`target_matrix`	A square numeric matrix of size `n` equal to the order of the graphs specifying a target matrix towards which the initial doubly stochastic matrix is shrunk each time the graph matching algorithm fails to provide a good minimum. Defaults to `NULL` in which case the target matrix is automatically chosen between the identity matrix or the uniform matrix on the n-simplex.

Value

A matrix of dimension $(n_1 + n_2) \times (n_1 + n_2)$ containing the distances between all the elements of the two samples put together.

Examples

```
gnp_params <- list(n = 24L, p = 1/3)
degree_params <- list(out_degree = rep(2, 24L), method = "configuration")
x <- nvd(sample_size = 10L, model = "gnp", !!!gnp_params)
y <- nvd(sample_size = 10L, model = "degree", !!!degree_params)
dist_nvd(x, y, "adjacency", "spectral")
```

`edge_count_global_variables`

Transform distance matrix in edge properties of minimal spanning tree

---

Description

Transform distance matrix in edge properties of minimal spanning tree

Usage

```
edge_count_global_variables(d, n1, k = 1L)
```

Arguments

`d`	A matrix of dimension $(n1 + n2)x(n1 + n2)$ containing the distances between all the elements of the two samples put together.
`n1`	An integer giving the size of the first sample.
`k`	An integer specifying the density of the minimal spanning tree to generate.

Value

A list of edge properties of the minimal spanning tree.

Examples

```
n1 <- 30L
n2 <- 10L
gnp_params <- list(n = 24L, p = 1/3)
degree_params <- list(out_degree = rep(2, 24L), method = "configuration")
x <- nvd(sample_size = n1, model = "gnp", !!!gnp_params)
y <- nvd(sample_size = n2, model = "degree", !!!degree_params)
d <- dist_nvd(x, y, representation = "laplacian", distance = "frobenius")
e <- edge_count_global_variables(d, n1, k = 5L)
```

generate_sigma_algebra *Sigma-Algebra generated by a Partition*

Description

Sigma-Algebra generated by a Partition

Usage

```
generate_sigma_algebra(x)
```

Arguments

x	Input partition stored as a `vertex_partition` object.

Value

Sigma-algebra

Examples

```
g <- igraph::make_ring(7)
m <- as.integer(c(1, 2, 1, 3, 4, 4, 3))
p <- as_vertex_partition(m)
sa <- generate_sigma_algebra(p)
all_full  <- purrr::modify_depth(sa, 2, ~ subgraph_full (g, .x))
all_intra <- purrr::modify_depth(sa, 2, ~ subgraph_intra(g, .x))
all_inter <- purrr::modify_depth(sa, 2, ~ subgraph_inter(g, .x))
```

inner-products *Inner-Products Between Networks*

Description

This is a collection of functions computing the inner product between two networks.

Usage

```
ipro_frobenius(x, y, representation = "laplacian")
```

Arguments

x	An `igraph` object or a matrix representing an underlying network.
y	An `igraph` object or a matrix representing an underlying network. Should have the same number of vertices as `x`.
representation	A string specifying the desired type of representation, among: `"adjacency"`, `"laplacian"`, `"modularity"` or `"graphon"`. Default is `"laplacian"`.

Value

A scalar measuring the angle between the two input networks.

Examples

```
g1 <- igraph::sample_gnp(20, 0.1)
g2 <- igraph::sample_gnp(20, 0.2)
ipro_frobenius(g1, g2, "adjacency")
```

`mean.nvd` *Fréchet Mean of Network-Valued Data*

Description

This function computes the sample Fréchet mean from an observed sample of network-valued random variables according to a specified matrix representation. It currently only supports the Euclidean geometry i.e. the sample Fréchet mean is obtained as the argmin of the sum of squared Frobenius distances.

Usage

```
## S3 method for class 'nvd'
mean(x, weights = rep(1, length(x)), representation = "adjacency", ...)
```

Arguments

`x`	An `nvd` object.
`weights`	A numeric vector specifying weights for each observation (default: equally weighted).
`representation`	A string specifying the graph representation to be used. Choices are adjacency, laplacian, modularity, graphon. Default is adjacency.
`...`	Other argument to be parsed to the `mean` function.

Value

The mean network in the chosen matrix representation assuming Euclidean geometry for now.

Examples

```
params <- list(n = 24L, p = 1/3)
x <- nvd(sample_size = 1L, model = "gnp", !!!params)
mean(x)
```

nvd *Network-Valued Data Constructor*

Description

This is the constructor for objects of class `nvd`.

Usage

```
nvd(sample_size, model, ...)
```

Arguments

`sample_size`	An integer specifying the sample size.
`model`	A string specifying the model to be used for sampling networks. All `tidygraph::play_` functions are supported. The model name corresponds to the name of the function without the `play_` prefix.
`...`	Model parameters to be passed to the model function.

Value

A `nvd` object which is a list of `tidygraph::tbl_graph` objects.

Examples

```
params <- list(n_dim = 1L, dim_size = 4L, order = 4L, p_rewire = 0.15)
nvd(sample_size = 1L, model = "smallworld", !!!params)
```

nvd-plot *MDS Visualization of Network Distributions*

Description

This function generates 2-dimensional plots of samples of networks via multi-dimensional scaling using all representations and distances included in the package.

Usage

```
## S3 method for class 'nvd'
autoplot(object, memberships = rep(1, length(object)), method = "mds", ...)

## S3 method for class 'nvd'
plot(x, method = "mds", ...)
```

Arguments

`object, x`	A list containing two samples of network-valued data stored as objects of class `nvd`.
`memberships`	An integer vector specifying the membership of each network to a specific sample. Defaults to `rep(1, length(nvd))` which assumes that all networks in the input `nvd` object belong to a single group.
`method`	A string specifying which dimensionality reduction method to use for projecting the samples into the cartesian plane. Choices are `"mds"`, `"tsne"` or `"umap"`. Defaults to `"mds"`.
`...`	Extra arguments to be passed to the plot function.

Value

Invisibly returns a `ggplot` object. In particular, the data set computed to generate the plot can be retrieved via `$data`. This is a `tibble` containing the following variables:

- `V1`: the x-coordinate of each observation in the plane,
- `V2`: the y-coordinate of each observation in the plane,
- `Label`: the sample membership of each observation,
- `Representation`: the type of matrix representation used to manipulate each observation,
- `Distance`: the distance used to measure how far each observation is from the others.

Examples

```
gnp_params <- list(n = 24L, p = 1/3)
degree_params <- list(out_degree = rep(2, 24L), method = "configuration")
x <- nvd(sample_size = 10L, model = "gnp", !!!gnp_params)
y <- nvd(sample_size = 10L, model = "degree", !!!degree_params)
mb <- c(rep(1, length(x)), rep(2, length(y)))
z <- as_nvd(c(x, y))
ggplot2::autoplot(z, memberships = mb)
plot(z, memberships = mb)
```

`power2` *Power Simulations for Permutation Tests*

Description

This function provides a Monte-Carlo estimate of the power of the permutation tests proposed in this package.

Usage

```
power2(
  sample_size1,
  model1,
  params1,
  sample_size2,
  model2,
  params2,
  representation = "adjacency",
  distance = "frobenius",
  stats = c("flipr:t_ip", "flipr:f_ip"),
  B = 1000L,
  alpha = 0.05,
  test = "exact",
  k = 5L,
  R = 1000L
)
```

Arguments

`sample_size1`	An integer specifying the size of the first sample.
`model1`	A string specifying the model to be used for sampling networks in the first sample. All `tidygraph::play_` functions are supported. The model name corresponds to the name of the function without the `play_` prefix.
`params1`	A list specifying the parameters to be passed to the model function that will generate the first sample.
`sample_size2`	An integer specifying the size of the second sample.
`model2`	A string specifying the model to be used for sampling networks in the second sample. All `tidygraph::play_` functions are supported. The model name corresponds to the name of the function without the `play_` prefix.
`params2`	A list specifying the parameters to be passed to the model
`representation`	A string specifying the desired type of representation, among: `"adjacency"`, `"laplacian"` and `"modularity"`. Defaults to `"adjacency"`.
`distance`	A string specifying the chosen distance for calculating the test statistic, among: `"hamming"`, `"frobenius"`, `"spectral"` and `"root-euclidean"`. Defaults to `"frobenius"`.
`stats`	A character vector specifying the chosen test statistic(s), among: `"original_edge_count"`, `"generalized_edge_count"`, `"weighted_edge_count"`, `"student_euclidean"`, `"welch_euclidean"` or any statistics based on inter-point distances available in the **flipr** package: `"flipr:student_ip"`, `"flipr:fisher_ip"`, `"flipr:bg_ip"`, `"flipr:energy_ip"`, `"flipr:cq_ip"`. Defaults to `c("flipr:student_ip", "flipr:fisher_ip")`.
`B`	The number of permutation or the tolerance. If this number is lower than `1`, it is intended as a tolerance. Otherwise, it is intended as the number of required permutations. Defaults to `1000L`.

`alpha`	Significance level for hypothesis testing. Defaults to `0.05`.
`test`	A character string specifying the formula to be used to compute the permutation p-value. Choices are `"estimate"`, `"upper_bound"` and `"exact"`. Defaults to `"exact"` which provides exact tests.
`k`	An integer specifying the density of the minimum spanning tree used for the edge count statistics. Defaults to `5L`.
`R`	Number of Monte-Carlo trials used to estimate the power. Defaults to `1000L`.

Details

Currently, six scenarios of pairs of populations are implemented. Scenario 0 allows to make sure that all our permutation tests are exact.

Value

A numeric value estimating the power of the test.

Examples

```
gnp_params <- list(n = 24L, p = 1/3)
degree_params <- list(out_degree = rep(2, 24L), method = "configuration")
power2(
  sample_size1 = 10L, model1 = "gnp", params1 = gnp_params,
  sample_size2 = 10L, model2 = "degree", params2 = degree_params,
  R = 10L,
  B = 100L
)
```

`push_to_graph_space` *Graph Sample Embeddings*

Description

A collection of functions to embed a sample of graphs into suitable spaces for further statistical analysis.

Usage

```
push_to_graph_space(obj)
```

Arguments

`obj`	An object of class `nvd` containing the sample of graphs.

Value

An object of class `nvd` containing the sample of graphs in graph space.

Examples

```
x <- nvd(sample_size = 5L, model = "gnp", n = 24L, p = 1/3)
x <- push_to_graph_space(x)
```

---

representations *Network Representation Functions*

Description

This is a collection of functions that convert a graph stored as an igraph object into a desired matrix representation among adjacency matrix, graph laplacian, modularity matrix or graphon (edge probability matrix).

Usage

```
repr_adjacency(network, validate = TRUE)

repr_laplacian(network, validate = TRUE)

repr_modularity(network, validate = TRUE)

repr_graphon(network, validate = TRUE)
```

Arguments

`network`	An igraph object.
`validate`	A boolean specifying whether the function should check the class of its input (default: `TRUE`).

Value

A numeric square matrix giving the desired network representation recorded in the object's class.

Examples

```
g <- igraph::sample_smallworld(1, 25, 3, 0.05)
repr_adjacency(g)
repr_laplacian(g)
repr_modularity(g)
repr_graphon(g)
```

repr_nvd — *Network-Valued to Matrix-Valued Data*

Description

Network-Valued to Matrix-Valued Data

Usage

```
repr_nvd(x, y = NULL, representation = "adjacency")
```

Arguments

`x`	An `nvd` object.
`y`	An `nvd` object. If `NULL` (default), it is not taken into account.
`representation`	A string specifying the requested matrix representation. Choices are: `"adjacency"`, `"laplacian"` or `"modularity"`. Default is `"adjacency"`.

Value

A list of matrices.

Examples

```
params <- list(n = 24L, p = 1/3)
x <- nvd(sample_size = 1L, model = "gnp", !!!params)
xm <- repr_nvd(x)
```

sample2_sbm — *Two-Sample Stochastic Block Model Generator*

Description

This function generates two samples of networks according to the stochastic block model (SBM). This is essentially a wrapper around `sample_sbm` which allows to sample a single network from the SBM.

Usage

```
sample2_sbm(n, nv, p1, b1, p2 = p1, b2 = b1, seed = NULL)
```

Arguments

`n`	Integer scalar giving the sample size.
`nv`	Integer scalar giving the number of vertices of the generated networks, common to all networks in both samples.
`p1`	The matrix giving the Bernoulli rates for the 1st sample. This is a KxK matrix, where K is the number of groups. The probability of creating an edge between vertices from groups i and j is given by element (i,j). For undirected graphs, this matrix must be symmetric.
`b1`	Numeric vector giving the number of vertices in each group for the first sample. The sum of the vector must match the number of vertices.
`p2`	The matrix giving the Bernoulli rates for the 2nd sample (default: same as 1st sample). This is a KxK matrix, where K is the number of groups. The probability of creating an edge between vertices from groups i and j is given by element (i,j). For undirected graphs, this matrix must be symmetric.
`b2`	Numeric vector giving the number of vertices in each group for the second sample (default: same as 1st sample). The sum of the vector must match the number of vertices.
`seed`	The seed for the random number generator (default: `NULL`).

Value

A length-2 list containing the two samples stored as `nvd` objects.

Examples

```
n <- 10
p1 <- matrix(
  data = c(0.1, 0.4, 0.1, 0.4,
           0.4, 0.4, 0.1, 0.4,
           0.1, 0.1, 0.4, 0.4,
           0.4, 0.4, 0.4, 0.4),
  nrow = 4,
  ncol = 4,
  byrow = TRUE
)
p2 <- matrix(
  data = c(0.1, 0.4, 0.4, 0.4,
           0.4, 0.4, 0.4, 0.4,
           0.4, 0.4, 0.1, 0.1,
           0.4, 0.4, 0.1, 0.4),
  nrow = 4,
  ncol = 4,
  byrow = TRUE
)
sim <- sample2_sbm(n, 68, p1, c(17, 17, 17, 17), p2, seed = 1234)
```

samplers *Graph samplers using edge distributions*

Description

A collection of functions to generate random graphs with specified edge distribution.

Usage

```
play_poisson(num_vertices, lambda = 1, directed = FALSE, loops = FALSE)

rpois_network(n, num_vertices, lambda = 1, directed = FALSE, loops = FALSE)

play_exponential(num_vertices, rate = 1, directed = FALSE, loops = FALSE)

rexp_network(n, num_vertices, rate = 1, directed = FALSE, loops = FALSE)

play_binomial(
  num_vertices,
  size = 1,
  prob = 0.5,
  directed = FALSE,
  loops = FALSE
)

rbinom_network(
  n,
  num_vertices,
  size = 1,
  prob = 0.5,
  directed = FALSE,
  loops = FALSE
)
```

Arguments

num_vertices	Number of vertices.
lambda	The mean parameter for the Poisson distribution (default: 1).
directed	A boolean specifying whether to generate directed or undirected graphs. Defaults to FALSE.
loops	A boolean specifying whether loops are authorized. Defaults to FALSE.
n	Sample size.
rate	The rate parameter for the exponential distribution (default: 1).
size	The number of trials for the binomial distribution (default: 1).
prob	The probability of success on each trial for the binomial distribution (default: 0.5).

Value

A object of class nvd containing the sample of graphs.

Examples

```
nvd <- rexp_network(10, 68)
```

---

statistics — *Test Statistics for Network Populations*

Description

This is a collection of functions that provide statistics for testing equality in distribution between samples of networks.

Usage

```
stat_student_euclidean(d, indices, ...)

stat_welch_euclidean(d, indices, ...)

stat_original_edge_count(d, indices, edge_count_prep, ...)

stat_generalized_edge_count(d, indices, edge_count_prep, ...)

stat_weighted_edge_count(d, indices, edge_count_prep, ...)
```

Arguments

`d`	Either a matrix of dimension $(n1 + n2)x(n1 + n2)$ containing the distances between all the elements of the two samples put together (for distance-based statistics) or the concatenation of the lists of matrix representations of networks in samples 1 and 2 for Euclidean t-Statistics.
`indices`	A vector of dimension $n1$ containing the indices of the elements of the first sample.
`...`	Extra parameters specific to some statistics.
`edge_count_prep`	A list of preprocessed data information used by edge count statistics and produced by `edge_count_global_variables`.

Details

In details, there are three main categories of statistics:

- *Euclidean t-Statistics*: both Student `stat_student_euclidean` version for equal variances and Welch `stat_welch_euclidean` version for unequal variances,
- *Statistics based on similarity graphs*: 3 types of edge count statistics.

Value

A scalar giving the value of the desired test statistic.

Examples

```
n1 <- 30L
n2 <- 10L
gnp_params <- list(n = 24L, p = 1/3)
degree_params <- list(out_degree = rep(2, 24L), method = "configuration")
x <- nvd(sample_size = n1, model = "gnp", !!!gnp_params)
y <- nvd(sample_size = n2, model = "degree", !!!degree_params)
r <- repr_nvd(x, y, representation = "laplacian")
stat_student_euclidean(r, 1:n1)
stat_welch_euclidean(r, 1:n1)
d <- dist_nvd(x, y, representation = "laplacian", distance = "frobenius")
ecp <- edge_count_global_variables(d, n1, k = 5L)
stat_original_edge_count(d, 1:n1, edge_count_prep = ecp)
stat_generalized_edge_count(d, 1:n1, edge_count_prep = ecp)
stat_weighted_edge_count(d, 1:n1, edge_count_prep = ecp)
```

`subgraphs` *Full, intra and inter subgraph generators*

Description

This is a collection of functions for extracting full, intra and inter subgraphs of a graph given a list of vertex subsets.

Usage

```
subgraph_full(g, vids)

subgraph_intra(g, vids)

subgraph_inter(g, vids)
```

Arguments

`g`	An `igraph` object.
`vids`	A list of integer vectors identifying vertex subsets.

Value

An `igraph` object storing a subgraph of type full, intra or inter.

Examples

```
g <- igraph::make_ring(10)
g_full  <- subgraph_full (g, list(1:3, 4:5, 8:10))
g_intra <- subgraph_intra(g, list(1:3, 4:5, 8:10))
g_inter <- subgraph_inter(g, list(1:3, 4:5, 8:10))
```

`test2_global` *Global Two-Sample Test for Network-Valued Data*

Description

This function carries out an hypothesis test where the null hypothesis is that the two populations of networks share the same underlying probabilistic distribution against the alternative hypothesis that the two populations come from different distributions. The test is performed in a non-parametric fashion using a permutational framework in which several statistics can be used, together with several choices of network matrix representations and distances between networks.

Usage

```
test2_global(
  x,
  y,
  representation = c("adjacency", "laplacian", "modularity", "transitivity"),
  distance = c("frobenius", "hamming", "spectral", "root-euclidean"),
  stats = c("flipr:t_ip", "flipr:f_ip"),
  B = 1000L,
  test = "exact",
  k = 5L,
  seed = NULL,
  ...
)
```

Arguments

`x`	Either an object of class nvd listing networks in sample 1 or a distance matrix of size $n_1 + n_2$.
`y`	Either an object of class nvd listing networks in sample 2 or an integer value specifying the size of sample 1 or an integer vector specifying the indices of the observations belonging to sample 1.
`representation`	A string specifying the desired type of representation, among: `"adjacency"`, `"laplacian"` and `"modularity"`. Defaults to `"adjacency"`.
`distance`	A string specifying the chosen distance for calculating the test statistic, among: `"hamming"`, `"frobenius"`, `"spectral"` and `"root-euclidean"`. Defaults to `"frobenius"`.

`stats`	A character vector specifying the chosen test statistic(s), among: `"original_edge_count"`, `"generalized_edge_count"`, `"weighted_edge_count"`, `"student_euclidean"`, `"welch_euclidean"` or any statistics based on inter-point distances available in the **flipr** package: `"flipr:student_ip"`, `"flipr:fisher_ip"`, `"flipr:bg_ip"`, `"flipr:energy_ip"`, `"flipr:cq_ip"`. Defaults to `c("flipr:student_ip", "flipr:fisher_ip")`.
`B`	The number of permutation or the tolerance. If this number is lower than `1`, it is intended as a tolerance. Otherwise, it is intended as the number of required permutations. Defaults to `1000L`.
`test`	A character string specifying the formula to be used to compute the permutation p-value. Choices are `"estimate"`, `"upper_bound"` and `"exact"`. Defaults to `"exact"` which provides exact tests.
`k`	An integer specifying the density of the minimum spanning tree used for the edge count statistics. Defaults to `5L`.
`seed`	An integer for specifying the seed of the random generator for result reproducibility. Defaults to `NULL`.
`...`	Extra arguments to be passed to the distance function.

Value

A `list` with three components: the value of the statistic for the original two samples, the p-value of the resulting permutation test and a numeric vector storing the values of the permuted statistics.

Examples

```
n <- 5L
gnp_params <- list(n = 24L, p = 1/3)
degree_params <- list(out_degree = rep(2, 24L), method = "configuration")

# Two different models for the two populations
x <- nvd(sample_size = n, model = "gnp", !!!gnp_params)
y <- nvd(sample_size = n, model = "degree", !!!degree_params)
t1 <- test2_global(x, y, representation = "modularity")
t1$pvalue

# Same model for the two populations
x <- nvd(sample_size = n, model = "gnp", !!!gnp_params)
y <- nvd(sample_size = n, model = "gnp", !!!gnp_params)
t2 <- test2_global(x, y, representation = "modularity")
t2$pvalue
```

`test2_local` *Local Two-Sample Test for Network-Valued Data*

Description

Local Two-Sample Test for Network-Valued Data

Usage

```
test2_local(
  x,
  y,
  partition,
  representation = "adjacency",
  distance = "frobenius",
  stats = c("flipr:t_ip", "flipr:f_ip"),
  B = 1000L,
  alpha = 0.05,
  test = "exact",
  k = 5L,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`x`	Either an object of class nvd listing networks in sample 1 or a distance matrix of size $n_1 + n_2$.
`y`	Either an object of class nvd listing networks in sample 2 or an integer value specifying the size of sample 1 or an integer vector specifying the indices of the observations belonging to sample 1.
`partition`	Either a list or an integer vector specifying vertex memberships into partition elements.
`representation`	A string specifying the desired type of representation, among: `"adjacency"`, `"laplacian"` and `"modularity"`. Defaults to `"adjacency"`.
`distance`	A string specifying the chosen distance for calculating the test statistic, among: `"hamming"`, `"frobenius"`, `"spectral"` and `"root-euclidean"`. Defaults to `"frobenius"`.
`stats`	A character vector specifying the chosen test statistic(s), among: `"original_edge_count"`, `"generalized_edge_count"`, `"weighted_edge_count"`, `"student_euclidean"`, `"welch_euclidean"` or any statistics based on inter-point distances available in the **flipr** package: `"flipr:student_ip"`, `"flipr:fisher_ip"`, `"flipr:bg_ip"`, `"flipr:energy_ip"`, `"flipr:cq_ip"`. Defaults to `c("flipr:student_ip", "flipr:fisher_ip")`.
`B`	The number of permutation or the tolerance. If this number is lower than `1`, it is intended as a tolerance. Otherwise, it is intended as the number of required permutations. Defaults to `1000L`.
`alpha`	Significance level for hypothesis testing. If set to 1, the function outputs properly adjusted p-values. If lower than 1, then only p-values lower than alpha are properly adjusted. Defaults to `0.05`.
`test`	A character string specifying the formula to be used to compute the permutation p-value. Choices are `"estimate"`, `"upper_bound"` and `"exact"`. Defaults to `"exact"` which provides exact tests.

`k`	An integer specifying the density of the minimum spanning tree used for the edge count statistics. Defaults to `5L`.
`seed`	An integer for specifying the seed of the random generator for result reproducibility. Defaults to `NULL`.
`verbose`	Boolean specifying whether information on intermediate tests should be printed in the process (default: `FALSE`).

Value

A length-2 list reporting the adjusted p-values of each element of the partition for the intra- and inter-tests.

Examples

```
n <- 5L
p1 <- matrix(
  data = c(0.1, 0.4, 0.1, 0.4,
           0.4, 0.4, 0.1, 0.4,
           0.1, 0.1, 0.4, 0.4,
           0.4, 0.4, 0.4, 0.4),
  nrow = 4,
  ncol = 4,
  byrow = TRUE
)
p2 <- matrix(
  data = c(0.1, 0.4, 0.4, 0.4,
           0.4, 0.4, 0.4, 0.4,
           0.4, 0.4, 0.1, 0.1,
           0.4, 0.4, 0.1, 0.4),
  nrow = 4,
  ncol = 4,
  byrow = TRUE
)
sim <- sample2_sbm(n, 68, p1, c(17, 17, 17, 17), p2, seed = 1234)
m <- as.integer(c(rep(1, 17), rep(2, 17), rep(3, 17), rep(4, 17)))
test2_local(sim$x, sim$y, m,
            seed = 1234,
            alpha = 0.05,
            B = 19)
```

`var2_nvd` *Fréchet Variance of Network-Valued Data from Inter-Point Distances*

Description

This function computes the Fréchet variance using exclusively inter-point distances. As such, it can accommodate any pair of representation and distance.

Usage

```
var2_nvd(x, representation = "adjacency", distance = "frobenius")
```

Arguments

`x`	An nvd object listing a sample of networks.
`representation`	A string specifying the graph representation to be used. Choices are adjacency, laplacian, modularity, graphon. Default is adjacency.
`distance`	A string specifying the distance to be used. Possible choices are: hamming, frobenius, spectral or root-euclidean. Default is frobenius.

Value

A positive scalar value evaluating the variance based on inter-point distances.

Examples

```
params <- list(n = 24L, p = 1/3)
x <- nvd(sample_size = 1L, model = "gnp", !!!params)
var2_nvd(x = x, representation = "graphon", distance = "frobenius")
```

`var_nvd` *Fréchet Variance of Network-Valued Data Around a Given Network*

Description

This function computes the Fréchet variance around a specified network from an observed sample of network-valued random variables according to a specified distance. In most cases, the user is willing to compute the sample variance, in which case the Fréchet variance has to be evaluated w.r.t. the sample Fréchet mean. In this case, it is important that the user indicates the same distance as the one (s)he used to separately compute the sample Fréchet mean. This function can also be used as is as the function to be minimized in order to find the Fréchet mean for a given distance.

Usage

```
var_nvd(x, x0, weights = rep(1, length(x)), distance = "frobenius")
```

Arguments

`x`	An nvd object listing a sample of networks.
`x0`	A network already in matrix representation around which to calculate variance (usually the Fréchet mean but not necessarily). Note that the chosen matrix representation is extracted from this parameter.
`weights`	A numeric vector specifying weights for each observation (default: equally weighted).

`distance`	A string specifying the distance to be used. Possible choices are: hamming, frobenius, spectral or root-euclidean. Default is frobenius. When the Fréchet mean is used as `x0` parameter, the distance should match the one used to compute the mean. This is not currently checked.

Value

A positive scalar value evaluating the amount of variability of the sample around the specified network.

Examples

```
params <- list(n = 24L, p = 1/3)
x <- nvd(sample_size = 1L, model = "gnp", !!!params)
m <- mean(x)
var_nvd(x = x, x0 = m, distance = "frobenius")
```

Index