Package 'bioregion'

January 31, 2025

Type Package

Title Comparison of Bioregionalisation Methods

Version 1.2.0

Description The main purpose of this package is to propose a transparent methodological framework to compare bioregionalisation methods based on hierarchical and non-hierarchical clustering algorithms (Kreft & Jetz (2010) <doi:10.1111/j.1365-2699.2010.02375.x>) and network algorithms (Lenor-
mand et al. (2019) <doi:10.1002/ece3.4718> and Leroy et al. (2019) <doi:10.1111/jbi.13674>).

Depends R (>= 4.0.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports ape, apcluster, bipartite, cluster, data.table, dbscan, dynamicTreeCut, fastcluster, fastkmedoids, ggplot2, grDevices, httr, igraph, mathjaxr, Matrix, phangorn, Rdpack, rlang, rmarkdown, segmented, sf, stats, tidyr, utils

RdMacros mathjaxr, Rdpack

LinkingTo Rcpp

Suggests ade4, dplyr, knitr, microbenchmark, rnaturalearth, rnaturalearthdata, testthat (>= 3.0.0)

VignetteBuilder knitr

RoxygenNote 7.3.2

URL `https://github.com/bioRgeo/bioregion`,
`https://bioRgeo.github.io/bioregion/`

BugReports `https://github.com/bioRgeo/bioregion/issues`

Config/testthat/edition 3

NeedsCompilation yes

Author Maxime Lenormand [aut, cre] (<`https://orcid.org/0000-0001-6362-3473`>),
Boris Leroy [aut] (<`https://orcid.org/0000-0002-7686-4302`>),
Pierre Denelle [aut] (<`https://orcid.org/0000-0001-5037-2281`>)

Maintainer Maxime Lenormand <maxime.lenormand@inrae.fr>

Repository CRAN

Date/Publication 2025-01-31 16:40:02 UTC

Contents

`betapart_to_bioregion` *Convert betapart dissimilarity to bioregion dissimilarity*

Description

This function converts dissimilarity results produced by the betapart package (and packages using betapart, such as phyloregion) into a dissimilarity object compatible with the bioregion package. This function only converts object types to make them compatible with bioregion; it does not modify the beta-diversity values. This function allows the inclusion of phylogenetic beta diversity to compute bioregions with bioregion.

Usage

```
betapart_to_bioregion(betapart_result)
```

Arguments

`betapart_result`	An object produced by the betapart package (e.g., using the `beta.pair` function).

Value

A dissimilarity object of class `bioregion.pairwise.metric`, compatible with the bioregion package.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)

Examples

```
comat <- matrix(sample(0:1000, size = 50, replace = TRUE,
prob = 1 / 1:1001), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

## Not run:
beta_div <- betapart::beta.pair.abund(comat)
betapart_to_bioregion(beta_div)

## End(Not run)
```

bioregionalization_metrics

Calculate metrics for one or several bioregionalizations

Description

This function calculates metrics for one or several bioregionalizations, typically based on outputs from `netclu_`, `hclu_`, or `nhclu_` functions. Some metrics may require users to provide either a similarity or dissimilarity matrix, or the initial species-site table.

Usage

```
bioregionalization_metrics(
  bioregionalization,
  dissimilarity = NULL,
  dissimilarity_index = NULL,
  net = NULL,
  site_col = 1,
  species_col = 2,
  eval_metric = "all"
)
```

Arguments

`bioregionalization`	A `bioregion.clusters` object.
`dissimilarity`	A `dist` object or a `bioregion.pairwise.metric` object (output from `similarity_to_dissimilarity(` Required if `eval_metric` includes `"pc_distance"` and `tree` is not a `bioregion.hierar.tree` object.
`dissimilarity_index`	A `character` string indicating the dissimilarity (beta-diversity) index to use if dissimilarity is a `data.frame` with multiple dissimilarity indices.
`net`	The site-species network (i.e., bipartite network). Should be provided as a `data.frame` if `eval_metric` includes `"avg_endemism"` or `"tot_endemism"`.
`site_col`	The name or index of the column representing site nodes (i.e., primary nodes). Should be provided if `eval_metric` includes `"avg_endemism"` or `"tot_endemism"`.
`species_col`	The name or index of the column representing species nodes (i.e., feature nodes). Should be provided if `eval_metric` includes `"avg_endemism"` or `"tot_endemism"`.
`eval_metric`	A `character` vector or a single `character` string indicating the metric(s) to be calculated to assess the effect of different numbers of clusters. Available options are `"pc_distance"`, `"anosim"`, `"avg_endemism"`, or `"tot_endemism"`. If `"all"` is specified, all metrics will be calculated.

Details

Evaluation metrics:

- `pc_distance`: This metric, as used by Holt et al. (2013), is the ratio of the between-cluster sum of dissimilarities (beta-diversity) to the total sum of dissimilarities for the full dissimilarity matrix. It is calculated in two steps:
  - Compute the total sum of dissimilarities by summing all elements of the dissimilarity matrix.
  - Compute the between-cluster sum of dissimilarities by setting within-cluster dissimilarities to zero and summing the matrix. The `pc_distance` ratio is obtained by dividing the between-cluster sum of dissimilarities by the total sum of dissimilarities.
- `anosim`: This metric is the statistic used in the Analysis of Similarities, as described in Castro-Insua et al. (2018). It compares between-cluster and within-cluster dissimilarities. The statistic is computed as: R = (r_B - r_W) / (N (N-1) / 4), where r_B and r_W are the average ranks of between-cluster and within-cluster dissimilarities, respectively, and N is the total number of sites. Note: This function does not estimate significance; for significance testing, use vegan::anosim().
- `avg_endemism`: This metric is the average percentage of endemism in clusters, as recommended by Kreft & Jetz (2010). It is calculated as: End_mean = sum_i (E_i / S_i) / K, where E_i is the number of endemic species in cluster i, S_i is the number of species in cluster i, and K is the total number of clusters.
- `tot_endemism`: This metric is the total endemism across all clusters, as recommended by Kreft & Jetz (2010). It is calculated as: End_tot = E / C, where E is the total number of endemic species (i.e., species found in only one cluster) and C is the number of non-endemic species.

Value

A `list` of class `bioregion.bioregionalization.metrics` with two to three elements:

- `args`: Input arguments.
- `evaluation_df`: A `data.frame` containing the `eval_metric` values for all explored numbers of clusters.
- `endemism_results`: If endemism calculations are requested, a list with the endemism results for each bioregionalization.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)

References

Castro-Insua A, Gómez-Rodríguez C & Baselga A (2018) Dissimilarity measures affected by richness differences yield biased delimitations of biogeographic realms. *Nature Communications* 9, 9-11.

Holt BG, Lessard J, Borregaard MK, Fritz SA, Araújo MB, Dimitrov D, Fabre P, Graham CH, Graves GR, Jønsson Ka, Nogués-Bravo D, Wang Z, Whittaker RJ, Fjeldså J & Rahbek C (2013) An update of Wallace's zoogeographic regions of the world. *Science* 339, 74-78.

Kreft H & Jetz W (2010) A framework for delineating biogeographical regions based on species distributions. *Journal of Biogeography* 37, 2029-2053.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_1_hierarchical_clustering.html#optimaln`.

Associated functions: compare_bioregionalizations find_optimal_n

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

comnet <- mat_to_net(comat)

dissim <- dissimilarity(comat, metric = "all")

# User-defined number of clusters
tree1 <- hclu_hierarclust(dissim,
                          n_clust = 10:15,
                          index = "Simpson")
tree1

a <- bioregionalization_metrics(tree1,
                                dissimilarity = dissim,
                                net = comnet,
                                site_col = "Node1",
                                species_col = "Node2",
                                eval_metric = c("tot_endemism",
                                                "avg_endemism",
                                                "pc_distance",
                                                "anosim"))
a
```

bioregion_metrics *Calculate contribution metrics for bioregions*

Description

This function calculates the number of sites per bioregion, as well as the number of species these sites have, the number of endemic species, and the proportion of endemism.

Usage

```
bioregion_metrics(bioregionalization, comat, map = NULL, col_bioregion = NULL)
```

Arguments

`bioregionalization`	A `bioregion.clusters` object.
`comat`	A co-occurrence `matrix` with sites as rows and species as columns.
`map`	A spatial `sf data.frame` with sites and bioregions. It is the output of the function `map_bioregions`. `NULL` by default.
`col_bioregion`	An `integer` specifying the column position of the bioregion.

Details

Endemic species are species found only in the sites belonging to one bioregion.

Value

A `data.frame` with 5 columns, or 6 if spatial coherence is computed.

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a5_3_summary_metrics.html`.

Associated functions: site_species_metrics bioregionalization_metrics

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
clust <- netclu_louvain(net)

bioregion_metrics(bioregionalization = clust,
                  comat = comat)
```

compare_bioregionalizations

Compare cluster memberships among multiple bioregionalizations

Description

This function computes pairwise comparisons for several bioregionalizations, usually outputs from `netclu_`, `hclu_`, or `nhclu_` functions. It also provides the confusion matrix from pairwise comparisons, enabling the user to compute additional comparison metrics.

Usage

```
compare_bioregionalizations(
  bioregionalizations,
  indices = c("rand", "jaccard"),
  cor_frequency = FALSE,
  store_pairwise_membership = TRUE,
  store_confusion_matrix = TRUE
)
```

Arguments

`bioregionalizations`	A `data.frame` object where each row corresponds to a site, and each column to a bioregionalization.
`indices`	`NULL` or `character`. Indices to compute for the pairwise comparison of bioregionalizations. Currently available metrics are `"rand"` and `"jaccard"`.
`cor_frequency`	A `boolean`. If `TRUE`, computes the correlation between each bioregionalization and the total frequency of co-membership of items across all bioregionalizations. This is useful for identifying which bioregionalization(s) is(are) most representative of all computed bioregionalizations.
`store_pairwise_membership`	A `boolean`. If `TRUE`, stores the pairwise membership of items in the output object.
`store_confusion_matrix`	A `boolean`. If `TRUE`, stores the confusion matrices of pairwise bioregionalization comparisons in the output object.

Details

This function operates in two main steps:

1. Within each bioregionalization, the function compares all pairs of items and documents whether they are clustered together (`TRUE`) or separately (`FALSE`). For example, if site 1 and site 2 are clustered in the same cluster in bioregionalization 1, their pairwise membership `site1_site2` will be `TRUE`. This output is stored in the `pairwise_membership` slot if `store_pairwise_membership = TRUE`.

2. Across all bioregionalizations, the function compares their pairwise memberships to determine similarity. For each pair of bioregionalizations, it computes a confusion matrix with the following elements:

- `a`: Number of item pairs grouped in both bioregionalizations.
- `b`: Number of item pairs grouped in the first but not in the second bioregionalization.
- `c`: Number of item pairs grouped in the second but not in the first bioregionalization.
- `d`: Number of item pairs not grouped in either bioregionalization.

The confusion matrix is stored in `confusion_matrix` if `store_confusion_matrix = TRUE`.

Based on these confusion matrices, various indices can be computed to measure agreement among bioregionalizations. The currently implemented indices are:

- **Rand index**: `(a + d) / (a + b + c + d)` Measures agreement by considering both grouped and ungrouped item pairs.
- **Jaccard index**: `a / (a + b + c)` Measures agreement based only on grouped item pairs.

These indices are complementary: the Jaccard index evaluates clustering similarity, while the Rand index considers both clustering and separation. For example, if two bioregionalizations never group the same pairs, their Jaccard index will be 0, but their Rand index may be > 0 due to ungrouped pairs.

Users can compute additional indices manually using the list of confusion matrices.

To identify which bioregionalization is most representative of the others, the function can compute the correlation between the pairwise membership of each bioregionalization and the total frequency of pairwise membership across all bioregionalizations. This is enabled by setting `cor_frequency = TRUE`.

Value

A `list` containing 4 to 7 elements:

1. **args**: A `list` of user-provided arguments.
2. **inputs**: A `list` containing information on the input bioregionalizations, such as the number of items clustered.
3. **pairwise_membership** (optional): If `store_pairwise_membership = TRUE`, a `boolean matrix` where `TRUE` indicates two items are in the same cluster, and `FALSE` indicates they are not.
4. **freq_item_pw_membership**: A `numeric vector` containing the number of times each item pair is clustered together, corresponding to the sum of rows in `pairwise_membership`.
5. **bioregionalization_freq_cor** (optional): If `cor_frequency = TRUE`, a `numeric vector` of correlations between individual bioregionalizations and the total frequency of pairwise membership.
6. **confusion_matrix** (optional): If `store_confusion_matrix = TRUE`, a `list` of confusion matrices for each pair of bioregionalizations.
7. **bioregionalization_comparison**: A `data.frame` containing comparison results, where the first column indicates the bioregionalizations compared, and the remaining columns contain the requested `indices`.

Author(s)

Boris Leroy (<leroy.boris@gmail.com>)
Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_2_compare_bioregionalizations.html.

Associated functions: bioregionalization_metrics

Examples

```
# We here compare three different bioregionalizations
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "Simpson")
bioregion1 <- nhclu_kmeans(dissim, n_clust = 3, index = "Simpson")

net <- similarity(comat, metric = "Simpson")
bioregion2 <- netclu_greedy(net)
bioregion3 <- netclu_walktrap(net)

# Make one single data.frame with the bioregionalizations to compare
compare_df <- merge(bioregion1$clusters, bioregion2$clusters, by = "ID")
compare_df <- merge(compare_df, bioregion3$clusters, by = "ID")
colnames(compare_df) <- c("Site", "Hclu", "Greedy", "Walktrap")
rownames(compare_df) <- compare_df$Site
compare_df <- compare_df[, c("Hclu", "Greedy", "Walktrap")]

# Running the function
compare_bioregionalizations(compare_df)

# Find out which bioregionalizations are most representative
compare_bioregionalizations(compare_df,
                            cor_frequency = TRUE)
```

`cut_tree` *Cut a hierarchical tree*

Description

This function is designed to work on a hierarchical tree and cut it at user-selected heights. It works with outputs from either `hclu_hierarclust` or `hclust` objects. The function allows for cutting the tree based on the chosen number(s) of clusters or specified height(s). Additionally, it includes a procedure to automatically determine the cutting height for the requested number(s) of clusters.

Usage

```
cut_tree(
  tree,
  n_clust = NULL,
  cut_height = NULL,
  find_h = TRUE,
  h_max = 1,
  h_min = 0,
  dynamic_tree_cut = FALSE,
  dynamic_method = "tree",
  dynamic_minClusterSize = 5,
  dissimilarity = NULL,
  ...
)
```

Arguments

`tree`	A `bioregion.hierar.tree` or an `hclust` object.
`n_clust`	An `integer` vector or a single `integer` indicating the number of clusters to be obtained from the hierarchical tree, or the output from `bioregionalization_metrics()`. This should not be used concurrently with `cut_height`.
`cut_height`	A `numeric` vector specifying the height(s) at which the tree should be cut. This should not be used concurrently with `n_clust` or `optim_method`.
`find_h`	A `boolean` indicating whether the cutting height should be determined for the requested `n_clust`.
`h_max`	A `numeric` value indicating the maximum possible tree height for determining the cutting height when `find_h = TRUE`.
`h_min`	A `numeric` value specifying the minimum possible height in the tree for determining the cutting height when `find_h = TRUE`.
`dynamic_tree_cut`	A `boolean` indicating whether the dynamic tree cut method should be used. If `TRUE`, `n_clust` and `cut_height` are ignored.
`dynamic_method`	A `character` string specifying the method to be used for dynamically cutting the tree: either `"tree"` (clusters searched only within the tree) or `"hybrid"` (clusters searched in both the tree and the dissimilarity matrix).
`dynamic_minClusterSize`	An `integer` indicating the minimum cluster size for the dynamic tree cut method (see dynamicTreeCut::cutreeDynamic()).
`dissimilarity`	Relevant only if `dynamic_method = "hybrid"`. Provide the dissimilarity `data.frame` used to build the `tree`.
`...`	Additional arguments passed to dynamicTreeCut::cutreeDynamic() to customize the dynamic tree cut method.

Details

The function supports two main methods for cutting the tree. First, the tree can be cut at a uniform height (specified by `cut_height` or determined automatically for the requested `n_clust`). Second, the dynamic tree cut method (Langfelder et al., 2008) can be applied, which adapts to the shape of branches in the tree, cutting at varying heights based on cluster positions.

The dynamic tree cut method has two variants:

- The tree-based variant (`dynamic_method = "tree"`) uses a top-down approach, relying solely on the tree and the order of clustered objects.
- The hybrid variant (`dynamic_method = "hybrid"`) employs a bottom-up approach, leveraging both the tree and the dissimilarity matrix to identify clusters based on dissimilarity among sites. This approach is useful for detecting outliers within clusters.

Value

If `tree` is an output from `hclu_hierarclust()`, the same object is returned with updated content (i.e., `args` and `clusters`). If `tree` is an `hclust` object, a `data.frame` containing the clusters is returned.

Note

The `find_h` argument is ignored if `dynamic_tree_cut = TRUE`, as cutting heights cannot be determined in this case.

Author(s)

Pierre Denelle (<pierre.denelle@gmail.com>)
Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Boris Leroy (<leroy.boris@gmail.com>)

References

Langfelder P, Zhang B & Horvath S (2008) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. *BIOINFORMATICS* 24, 719-720.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_1_hierarchical_clustering.html`.

Associated functions: hclu_hierarclust

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site", 1:20)
colnames(comat) <- paste0("Species", 1:25)

simil <- similarity(comat, metric = "all")
```

```
dissimilarity <- similarity_to_dissimilarity(simil)

# User-defined number of clusters
tree1 <- hclu_hierarclust(dissimilarity,
                          n_clust = 5)
tree2 <- cut_tree(tree1, cut_height = .05)
tree3 <- cut_tree(tree1, n_clust = c(3, 5, 10))
tree4 <- cut_tree(tree1, cut_height = c(.05, .1, .15, .2, .25))
tree5 <- cut_tree(tree1, n_clust = c(3, 5, 10), find_h = FALSE)

hclust_tree <- tree2$algorithm$final.tree
clusters_2 <- cut_tree(hclust_tree, n_clust = 10)

cluster_dynamic <- cut_tree(tree1, dynamic_tree_cut = TRUE,
                            dissimilarity = dissimilarity)
```

`dissimilarity` *Compute dissimilarity metrics (beta-diversity) between sites based on species composition*

Description

This function generates a `data.frame` where each row provides one or several dissimilarity metrics between pairs of sites, based on a co-occurrence `matrix` with sites as rows and species as columns.

Usage

```
dissimilarity(comat, metric = "Simpson", formula = NULL, method = "prodmat")
```

Arguments

`comat`	A co-occurrence `matrix` with sites as rows and species as columns.
`metric`	A `character` vector or a single `character` string specifying the metrics to compute (see Details). Available options are `"abc"`, `"ABC"`, `"Jaccard"`, `"Jaccardturn"`, `"Sorensen"`, `"Simpson"`, `"Bray"`, `"Brayturn"`, and `"Euclidean"`. If `"all"` is specified, all metrics will be calculated. Can be set to `NULL` if `formula` is used.
`formula`	A `character` vector or a single `character` string specifying custom formula(s) based on the `a`, `b`, `c`, `A`, `B`, and `C` quantities (see Details). The default is `NULL`.
`method`	A `character` string specifying the method to compute `abc` (see Details). The default is `"prodmat"`, which is more efficient but memory-intensive. Alternatively, `"loops"` is less memory-intensive but slower.

Details

With a the number of species shared by a pair of sites, b species only present in the first site and c species only present in the second site.

Jaccard = (b + c) / (a + b + c)

Jaccardturn = 2min(b, c) / (a + 2min(b, c)) (Baselga, 2012)

Sorensen = (b + c) / (2a + b + c)

Simpson = min(b, c) / (a + min(b, c))

If abundances data are available, Bray-Curtis and its turnover component can also be computed with the following equation:

Bray = (B + C) / (2A + B + C)

Brayturn = min(B, C)/(A + min(B, C)) (Baselga, 2013)

with A the sum of the lesser values for common species shared by a pair of sites. B and C are the total number of specimens counted at both sites minus A.

`formula` can be used to compute customized metrics with the terms a, b, c, A, B, and C. For example `formula = c("pmin(b,c) / (a + pmin(b,c))", "(B + C) / (2*A + B + C)")` will compute the Simpson and Bray-Curtis dissimilarity metrics, respectively. Note that `pmin` is used in the Simpson formula because a, b, c, A, B and C are `numeric` vectors.

Euclidean computes the Euclidean distance between each pair of sites.

Value

A `data.frame` with the additional class `bioregion.pairwise.metric`, containing one or several dissimilarity metrics between pairs of sites. The first two columns represent the pairs of sites. There is one column per similarity metric provided in `metric` and `formula`, except for the `abc` and `ABC` metrics, which are stored in three separate columns (one for each letter).

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

References

Baselga, A. (2012) The Relationship between Species Replacement, Dissimilarity Derived from Nestedness, and Nestedness. *Global Ecology and Biogeography*, 21(12), 1223–1232.

Baselga, A. (2013) Separating the two components of abundance-based dissimilarity: balanced changes in abundance vs. abundance gradients. *Methods in Ecology and Evolution*, 4(6), 552–557.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a3_pairwise_metrics.html`.

Associated functions: similarity dissimilarity_to_similarity

Examples

```
comat <- matrix(sample(0:1000, size = 50, replace = TRUE,
prob = 1 / 1:1001), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

dissim <- dissimilarity(comat,
metric = c("abc", "ABC", "Simpson", "Brayturn"))

dissim <- dissimilarity(comat, metric = "all",
formula = "1 - (b + c) / (a + b + c)")
```

`dissimilarity_to_similarity` *Convert dissimilarity metrics to similarity metrics*

Description

This function converts a `data.frame` of dissimilarity metrics (beta diversity) between sites into similarity metrics.

Usage

```
dissimilarity_to_similarity(dissimilarity, include_formula = TRUE)
```

Arguments

`dissimilarity`	the output object from `dissimilarity()` or `similarity_to_dissimilarity()`.
`include_formula`	a `boolean` indicating whether metrics based on custom formula(s) should also be converted (see Details). The default is `TRUE`.

Value

A `data.frame` with the additional class `bioregion.pairwise.metric`, providing similarity metrics for each pair of sites based on a dissimilarity object.

Note

The behavior of this function changes depending on column names. Columns `Site1` and `Site2` are copied identically. If there are columns called `a`, `b`, `c`, `A`, `B`, `C` they will also be copied identically. If there are columns based on your own formula (argument `formula` in `dissimilarity()`) or not in the original list of dissimilarity metrics (argument `metrics` in `dissimilarity()`) and if the argument `include_formula` is set to `FALSE`, they will also be copied identically. Otherwise there are going to be converted like they other columns (default behavior).

If a column is called `Euclidean`, the similarity will be calculated based on the following formula:

Euclidean similarity = 1 / (1 - Euclidean distance)

Otherwise, all other columns will be transformed into dissimilarity with the following formula:

similarity = 1 - dissimilarity

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Boris Leroy (<leroy.boris@gmail.com>)
Pierre Denelle (<pierre.denelle@gmail.com>)

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a3_pairwise_metrics.html.

Associated functions: similarity dissimilarity_to_similarity

Examples

```
comat <- matrix(sample(0:1000, size = 50, replace = TRUE,
prob = 1 / 1:1001), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

dissimil <- dissimilarity(comat, metric = "all")
dissimil

similarity <- dissimilarity_to_similarity(dissimil)
similarity
```

---

find_optimal_n *Search for an optimal number of clusters in a list of bioregionalizations*

Description

This function aims to optimize one or several criteria on a set of ordered bioregionalizations. It is typically used to find one or more optimal cluster counts on hierarchical trees to cut or ranges of bioregionalizations from k-means or PAM. Users should exercise caution in other cases (e.g., unordered bioregionalizations or unrelated bioregionalizations).

Usage

```
find_optimal_n(
  bioregionalizations,
  metrics_to_use = "all",
  criterion = "elbow",
  step_quantile = 0.99,
  step_levels = NULL,
  step_round_above = TRUE,
  metric_cutoffs = c(0.5, 0.75, 0.9, 0.95, 0.99, 0.999),
  n_breakpoints = 1,
  plot = TRUE
)
```

Arguments

`bioregionalizations`
: A `bioregion.bioregionalization.metrics` object (output from `bioregionalization_metrics()`) or a `data.frame` with the first two columns named `K` (bioregionalization name) and `n_clusters` (number of clusters), followed by columns with numeric evaluation metrics.

`metrics_to_use`
: A `character` vector or single string specifying metrics in `bioregionalizations` for calculating optimal clusters. Defaults to `"all"` (uses all metrics).

`criterion`
: A `character` string specifying the criterion to identify optimal clusters. Options include `"elbow"`, `"increasing_step"`, `"decreasing_step"`, `"cutoff"`, `"breakpoints"`, `"min"`, or `"max"`. Defaults to `"elbow"`. See Details.

`step_quantile`
: For `"increasing_step"` or `"decreasing_step"`, specifies the quantile of differences between consecutive bioregionalizations as the cutoff to identify significant steps in `eval_metric`.

`step_levels`
: For `"increasing_step"` or `"decreasing_step"`, specifies the number of largest steps to retain as cutoffs.

`step_round_above`
: A `boolean` indicating whether the optimal clusters are above (`TRUE`) or below (`FALSE`) identified steps. Defaults to `TRUE`.

`metric_cutoffs`
: For `criterion = "cutoff"`, specifies the cutoffs of `eval_metric` to extract cluster counts.

`n_breakpoints`
: Specifies the number of breakpoints to find in the curve. Defaults to 1.

`plot`
: A `boolean` indicating if a plot of the first `eval_metric` with identified optimal clusters should be drawn.

Details

This function explores evaluation metric ~ cluster relationships, applying criteria to find optimal cluster counts.

Note on criteria: Several criteria can return multiple optimal cluster counts, emphasizing hierarchical or nested bioregionalizations. This approach aligns with modern recommendations for biological datasets, as seen in Ficetola et al. (2017)'s reanalysis of Holt et al. (2013).

Criteria for optimal clusters:

- `elbow`: Identifies the "elbow" point in the evaluation metric curve, where incremental improvements diminish. Based on a method to find the maximum distance from a straight line linking curve endpoints.

- `increasing_step` or `decreasing_step`: Highlights significant increases or decreases in metrics by analyzing pairwise differences between bioregionalizations. Users specify `step_quantile` or `step_levels`.
- `cutoffs`: Derives clusters from specified metric cutoffs, e.g., as in Holt et al. (2013). Adjust cutoffs based on spatial scale.
- `breakpoints`: Uses segmented regression to find breakpoints. Requires specifying `n_breakpoints`.
- `min` & `max`: Selects clusters at minimum or maximum metric values.

Value

A `list` of class `bioregion.optimal.n` with these elements:

- `args`: Input arguments.
- `evaluation_df`: The input evaluation `data.frame`, appended with `boolean` columns for optimal cluster counts.
- `optimal_nb_clusters`: A `list` with optimal cluster counts for each metric in `"metrics_to_use"`, based on the chosen `criterion`.
- `plot`: The plot (if requested).

Note

Please note that finding the optimal number of clusters is a procedure which normally requires decisions from the users, and as such can hardly be fully automatized. Users are strongly advised to read the references indicated below to look for guidance on how to choose their optimal number(s) of clusters. Consider the "optimal" numbers of clusters returned by this function as first approximation of the best numbers for your bioregionalization.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)

References

Holt BG, Lessard J, Borregaard MK, Fritz SA, Araújo MB, Dimitrov D, Fabre P, Graham CH, Graves GR, Jønsson Ka, Nogués-Bravo D, Wang Z, Whittaker RJ, Fjeldså J & Rahbek C (2013) An update of Wallace's zoogeographic regions of the world. *Science* 339, 74-78.

Ficetola GF, Mazel F & Thuiller W (2017) Global determinants of zoogeographical boundaries. *Nature Ecology & Evolution* 1, 0089.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_1_hierarchical_clustering.html#optimaln`.

Associated functions: hclu_hierarclust

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "all")

# User-defined number of clusters
tree <- hclu_hierarclust(dissim,
                          optimal_tree_method = "best",
                          n_clust = 5:10)
tree

a <- bioregionalization_metrics(tree,
                                dissimilarity = dissim,
                                species_col = "Node2",
                                site_col = "Node1",
                                eval_metric = "anosim")

find_optimal_n(a, criterion = 'increasing_step', plot = FALSE)
```

`fishdf` *Spatial distribution of fish in Europe (data.frame)*

Description

A dataset containing the abundance of 195 species in 338 sites.

Usage

```
fishdf
```

Format

A `data.frame` with 2,703 rows and 3 columns:

Site Unique site identifier (corresponding to the field ID of fishsf)

Species Unique species identifier

Abundance Species abundance

`fishmat` *Spatial distribution of fish in Europe (co-occurrence matrix)*

Description

A dataset containing the abundance of each of the 195 species in each of the 338 sites.

Usage

```
fishmat
```

Format

A co-occurrence `matrix` with sites as rows and species as columns. Each element of the matrix represents the abundance of the species in the site.

`fishsf` *Spatial distribution of fish in Europe*

Description

A dataset containing the geometry of the 338 sites.

Usage

```
fishsf
```

Format

A

ID Unique site identifier

geometry Geometry of the site

hclu_diana — *Divisive hierarchical clustering based on dissimilarity or beta-diversity*

Description

This function computes a divisive hierarchical clustering from a dissimilarity (beta-diversity) `data.frame`, calculates the cophenetic correlation coefficient, and can generate clusters from the tree if requested by the user. The function implements randomization of the dissimilarity matrix to generate the tree, with a selection method based on the optimal cophenetic correlation coefficient. Typically, the dissimilarity `data.frame` is a `bioregion.pairwise.metric` object obtained by running `similarity` or `similarity` followed by `similarity_to_dissimilarity`.

Usage

```
hclu_diana(
  dissimilarity,
  index = names(dissimilarity)[3],
  n_clust = NULL,
  cut_height = NULL,
  find_h = TRUE,
  h_max = 1,
  h_min = 0
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the remaining column(s) contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`n_clust`	An `integer` vector or a single `integer` indicating the number of clusters to be obtained from the hierarchical tree, or the output from bioregionalization_metrics. Should not be used concurrently with `cut_height`.
`cut_height`	A `numeric` vector indicating the height(s) at which the tree should be cut. Should not be used concurrently with `n_clust`.
`find_h`	A `boolean` indicating whether the cutting height should be determined for the requested `n_clust`.
`h_max`	A `numeric` value indicating the maximum possible tree height for the chosen `index`.
`h_min`	A `numeric` value indicating the minimum possible height in the tree for the chosen `index`.

Details

The function is based on diana. Chapter 6 of Kaufman & Rousseeuw (1990) fully details the functioning of the diana algorithm.

To find an optimal number of clusters, see `bioregionalization_metrics()`

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` describing the characteristics of the clustering process.
4. **algorithm**: A `list` containing all objects associated with the clustering procedure, such as the original cluster objects.
5. **clusters**: A `data.frame` containing the clustering results.

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

References

Kaufman L & Rousseeuw PJ (2009) Finding groups in data: An introduction to cluster analysis. In & Sons. JW (ed.), *Finding groups in data: An introduction to cluster analysis*.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_1_hierarchical_clustering.html`.

Associated functions: cut_tree

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "all")

data("fishmat")
fishdissim <- dissimilarity(fishmat)
fish_diana <- hclu_diana(fishdissim, index = "Simpson")
```

hclu_hierarclust *Hierarchical clustering based on dissimilarity or beta-diversity*

Description

This function generates a hierarchical tree from a dissimilarity (beta-diversity) `data.frame`, calculates the cophenetic correlation coefficient, and optionally retrieves clusters from the tree upon user request. The function includes a randomization process for the dissimilarity matrix to generate the tree, with two methods available for constructing the final tree. Typically, the dissimilarity `data.frame` is a `bioregion.pairwise.metric` object obtained by running `similarity`, or by running `similarity` followed by `similarity_to_dissimilarity`.

Usage

```
hclu_hierarclust(
  dissimilarity,
  index = names(dissimilarity)[3],
  method = "average",
  randomize = TRUE,
  n_runs = 100,
  keep_trials = FALSE,
  optimal_tree_method = "iterative_consensus_tree",
  n_clust = NULL,
  cut_height = NULL,
  find_h = TRUE,
  h_max = 1,
  h_min = 0,
  consensus_p = 0.5,
  verbose = TRUE
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the subsequent column(s) contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`method`	The name of the hierarchical classification method, as in hclust. Should be one of `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"` (= UPGMA), `"mcquitty"` (= WPGMA), `"median"` (= WPGMC), or `"centroid"` (= UPGMC).
`randomize`	A boolean indicating whether the dissimilarity matrix should be randomized to account for the order of sites in the dissimilarity matrix.
`n_runs`	The number of trials for randomizing the dissimilarity matrix.

`keep_trials`	A `boolean` indicating whether all random trial results should be stored in the output object. Set to `FALSE` to save space if your `dissimilarity` object is large. Note that this cannot be set to `TRUE` if `optimal_tree_method = "iterative_consensus_tree"`.
`optimal_tree_method`	A `character` string indicating how the final tree should be obtained from all trials. Possible values are `"iterative_consensus_tree"` (default), `"best"`, and `"consensus"`. **We recommend** `"iterative_consensus_tree"`**. See Details.**
`n_clust`	An `integer` vector or a single `integer` indicating the number of clusters to be obtained from the hierarchical tree, or the output from bioregionalization_metrics. This parameter should not be used simultaneously with `cut_height`.
`cut_height`	A `numeric` vector indicating the height(s) at which the tree should be cut. This parameter should not be used simultaneously with `n_clust`.
`find_h`	A `boolean` indicating whether the height of the cut should be found for the requested `n_clust`.
`h_max`	A `numeric` value indicating the maximum possible tree height for the chosen `index`.
`h_min`	A `numeric` value indicating the minimum possible height in the tree for the chosen `index`.
`consensus_p`	A `numeric` value (applicable only if `optimal_tree_method = "consensus"`) indicating the threshold proportion of trees that must support a region/cluster for it to be included in the final consensus tree.
`verbose`	A `boolean` (applicable only if `optimal_tree_method = "iterative_consensus_tree"`) indicating whether to display progress messages. Set to `FALSE` to suppress these messages.

Details

The function is based on hclust. The default method for the hierarchical tree is `average`, i.e. UPGMA as it has been recommended as the best method to generate a tree from beta diversity dissimilarity (Kreft & Jetz, 2010).

Clusters can be obtained by two methods:

- Specifying a desired number of clusters in `n_clust`
- Specifying one or several heights of cut in `cut_height`

To find an optimal number of clusters, see `bioregionalization_metrics()`

It is important to pay attention to the fact that the order of rows in the input distance matrix influences the tree topology as explained in Dapporto (2013). To address this, the function generates multiple trees by randomizing the distance matrix.

Two methods are available to obtain the final tree:

- `optimal_tree_method = "iterative_consensus_tree"`: The Iterative Hierarchical Consensus Tree (IHCT) method reconstructs a consensus tree by iteratively splitting the dataset into two subclusters based on the pairwise dissimilarity of sites across `n_runs` trees based on `n_runs` randomizations of the distance matrix. At each iteration, it identifies the majority membership of sites into two stable groups across all trees, calculates the height based on the selected linkage method (`method`), and enforces monotonic constraints on node heights to produce a coherent tree structure. This approach provides a robust, hierarchical representation of site relationships, balancing cluster stability and hierarchical constraints.

- `optimal_tree_method = "best"`: This method selects one tree among with the highest cophenetic correlation coefficient, representing the best fit between the hierarchical structure and the original distance matrix.
- `optimal_tree_method = "consensus"`: This method constructs a consensus tree using phylogenetic methods with the function consensus. When using this option, you must set the `consensus_p` parameter, which indicates the proportion of trees that must contain a region/cluster for it to be included in the final consensus tree. Consensus trees lack an inherent height because they represent a majority structure rather than an actual hierarchical clustering. To assign heights, we use a non-negative least squares method (nnls.tree) based on the initial distance matrix, ensuring that the consensus tree preserves approximate distances among clusters.

We recommend using the `"iterative_consensus_tree"` as all the branches of this tree will always reflect the majority decision among many randomized versions of the distance matrix. This method is inspired by Dapporto et al. (2015), which also used the majority decision among many randomized versions of the distance matrix, but it expands it to reconstruct the entire topology of the tree iteratively.

We do not recommend using the basic `consensus` method because in many contexts it provides inconsistent results, with a meaningless tree topology and a very low cophenetic correlation coefficient.

For a fast exploration of the tree, we recommend using the `best` method which will only select the tree with the highest cophenetic correlation coefficient among all randomized versions of the distance matrix.

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` describing the characteristics of the clustering process.
4. **algorithm**: A `list` containing all objects associated with the clustering procedure, such as the original cluster objects.
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, users can find the following elements:

- `trials`: A list containing all randomization trials. Each trial includes the dissimilarity matrix with randomized site order, the associated tree, and the cophenetic correlation coefficient (Spearman) for that tree.
- `final.tree`: An `hclust` object representing the final hierarchical tree to be used.
- `final.tree.coph.cor`: The cophenetic correlation coefficient between the initial dissimilarity matrix and the `final.tree`.

Author(s)

Boris Leroy (<leroy.boris@gmail.com>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Maxime Lenormand (<maxime.lenormand@inrae.fr>)

References

Kreft H & Jetz W (2010) A framework for delineating biogeographical regions based on species distributions. *Journal of Biogeography* 37, 2029-2053.

Dapporto L, Ramazzotti M, Fattorini S, Talavera G, Vila R & Dennis, RLH (2013) Recluster: an unbiased clustering procedure for beta-diversity turnover. *Ecography* 36, 1070–1075.

Dapporto L, Ciolli G, Dennis RLH, Fox R & Shreeve TG (2015) A new procedure for extrapolating turnover regionalization at mid-small spatial scales, tested on British butterflies. *Methods in Ecology and Evolution* 6 , 1287–1297.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_1_hierarchical_clustering.html.

Associated functions: cut_tree

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "Simpson")

# User-defined number of clusters
tree1 <- hclu_hierarclust(dissim,
                          n_clust = 5)
tree1
plot(tree1)
str(tree1)
tree1$clusters

# User-defined height cut
# Only one height
tree2 <- hclu_hierarclust(dissim,
                          cut_height = .05)
tree2
tree2$clusters

# Multiple heights
tree3 <- hclu_hierarclust(dissim,
                          cut_height = c(.05, .15, .25))

tree3$clusters # Mind the order of height cuts: from deep to shallow cuts
```

```
# Info on each partition can be found in table cluster_info
tree3$cluster_info
plot(tree3)
```

`hclu_optics` *OPTICS hierarchical clustering algorithm*

Description

This function performs semi-hierarchical clustering based on dissimilarity using the OPTICS algorithm (Ordering Points To Identify the Clustering Structure).

Usage

```
hclu_optics(
  dissimilarity,
  index = names(dissimilarity)[3],
  minPts = NULL,
  eps = NULL,
  xi = 0.05,
  minimum = FALSE,
  show_hierarchy = FALSE,
  algorithm_in_output = TRUE,
  ...
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the subsequent column(s) contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`minPts`	A `numeric` value specifying the minPts argument of dbscan. minPts is the minimum number of points required to form a dense region. By default, it is set to the natural logarithm of the number of sites in `dissimilarity`.
`eps`	A `numeric` value specifying the eps argument of optics. It defines the upper limit of the size of the epsilon neighborhood. Limiting the neighborhood size improves performance and has no or very little impact on the ordering as long as it is not set too low. If not specified (default behavior), the largest minPts-distance in the dataset is used, which gives the same result as infinity.
`xi`	A `numeric` value specifying the steepness threshold to identify clusters hierarchically using the Xi method (see optics).

`minimum`	A `boolean` specifying whether the hierarchy should be pruned from the output to only retain clusters at the "minimal" level, i.e., only leaf / non-overlapping clusters. If `TRUE`, then the argument `show_hierarchy` should be set to `FALSE`.
`show_hierarchy`	A `boolean` specifying whether the hierarchy of clusters should be included in the output. By default, the hierarchy is not visible in the clusters obtained from OPTICS; it can only be visualized by plotting the OPTICS object. If `show_hierarchy = TRUE`, the output cluster `data.frame` will contain additional columns showing the hierarchy of clusters.
`algorithm_in_output`	A `boolean` indicating whether the original output of dbscan should be returned in the output (`TRUE` by default, see Value).
`...`	Additional arguments to be passed to `optics()` (see optics).

Details

The OPTICS (Ordering points to identify the clustering structure) is a semi-hierarchical clustering algorithm which orders the points in the dataset such that points which are closest become neighbors, and calculates a reachability distance for each point. Then, clusters can be extracted in a hierarchical manner from this reachability distance, by identifying clusters depending on changes in the relative cluster density. The reachability plot should be explored to understand the clusters and their hierarchical nature, by running plot on the output of the function if `algorithm_in_output = TRUE`: `plot(object$algorithm)`. We recommend reading (Hahsler et al., 2019) to grasp the algorithm, how it works, and what the clusters mean.

To extract the clusters, we use the extractXi function which is based on the steepness of the reachability plot (see optics)

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` describing the characteristics of the clustering process.
4. **algorithm**: A `list` containing all objects associated with the clustering procedure, such as the original cluster objects.
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find the output of optics.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

References

Hahsler M, Piekenbrock M & Doran D (2019) Dbscan: Fast density-based clustering with R. *Journal of Statistical Software* 91, 1–30.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_1_hierarchical_clustering.html`.

Associated functions: nhclu_dbscan

Examples

```
dissim <- dissimilarity(fishmat, metric = "all")

clust1 <- hclu_optics(dissim, index = "Simpson")
clust1

# Visualize the optics plot (the hierarchy of clusters is illustrated at the
# bottom)
plot(clust1$algorithm)

# Extract the hierarchy of clusters
clust1 <- hclu_optics(dissim, index = "Simpson", show_hierarchy = TRUE)
clust1
```

`install_binaries` *Download, unzip, check permissions, and test the bioregion's binary files*

Description

This function downloads and unzips the 'bin' folder required to run certain functions of the `bioregion` package. It also verifies if the files have the necessary permissions to be executed as programs. Finally, it tests whether the binary files are running correctly.

Usage

```
install_binaries(
  binpath = "tempdir",
  download_only = FALSE,
  infomap_version = c("2.1.0", "2.6.0", "2.7.1", "2.8.0")
)
```

Arguments

`binpath`	A `character` string specifying the path to the folder that will host the `bin` folder containing the binary files (see Details).
`download_only`	A `logical` value indicating whether the function should only download the `bin.zip` file or perform the entire process (see Details).
`infomap_version`	A `character` vector or a single `character` string specifying the Infomap version(s) to install.

Details

By default, the binary files are installed in R's temporary directory (`binpath = "tempdir"`). In this case, the `bin` folder will be automatically removed at the end of the R session. Alternatively, the binary files can be installed in the `bioregion` package folder (`binpath = "pkgfolder"`).

A custom folder path can also be specified. In this case, and only in this case, `download_only` can be set to `TRUE`, but you must ensure that the files have the required permissions to be executed as programs.

In all cases, PLEASE MAKE SURE to update the `binpath` and `check_install` parameters accordingly in netclu_infomap, netclu_louvain, and netclu_oslom.

Value

No return value.

Note

Currently, only Infomap versions 2.1.0, 2.6.0, 2.7.1, and 2.8.0 are available.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Boris Leroy (<leroy.boris@gmail.com>)
Pierre Denelle (<pierre.denelle@gmail.com>)

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a1_install_binary_files.html.

---

map_bioregions *Create a map of bioregions*

Description

This plot function can be used to visualize bioregions based on a `bioregion.clusters` object combined with a geometry (`sf` objects).

Usage

```
map_bioregions(clusters, geometry, write_clusters = FALSE, plot = TRUE, ...)
```

Arguments

`clusters`	An object of class `bioregion.clusters` or a `data.frame`. If a `data.frame` is used, the first column should represent the sites' ID, and the subsequent column(s) should represent the clusters.
`geometry`	A spatial object that can be handled by the `sf` package. The first attribute should correspond to the sites' ID (see Details).
`write_clusters`	A `boolean` indicating if the `clusters` should be added to the `geometry`.
`plot`	A `boolean` indicating if the plot should be drawn.
`...`	Further arguments to be passed to `sf::plot()`.

Details

The `clusters` and `geometry` site IDs should correspond. They should have the same type (i.e., `character` if `clusters` is a `bioregion.clusters` object) and the sites of `clusters` should be included in the sites of `geometry`.

Value

One or several maps of bioregions if `plot = TRUE` and the geometry with additional clusters' attributes if `write_clusters = TRUE`.

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Boris Leroy (<`leroy.boris@gmail.com`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)

Examples

```
data(fishmat)
data(fishsf)

net <- similarity(fishmat, metric = "Simpson")
clu <- netclu_greedy(net)
map <- map_bioregions(clu, fishsf, write_clusters = TRUE, plot = FALSE)
```

`mat_to_net` *Create a data.frame from a contingency table*

Description

This function generates a two- or three-column `data.frame`, where each row represents the interaction between two nodes (e.g., site and species) and an optional third column indicates the weight of the interaction (if `weight = TRUE`). The input is a contingency table, with rows representing one set of entities (e.g., site) and columns representing another set (e.g., species).

Usage

```
mat_to_net(
  mat,
  weight = FALSE,
  remove_zeroes = TRUE,
  include_diag = TRUE,
  include_lower = TRUE
)
```

Arguments

`mat`	A contingency table (i.e., a `matrix`).
`weight`	A `logical` value indicating whether the values in the matrix should be interpreted as interaction weights.
`remove_zeroes`	A `logical` value determining whether interactions with a weight equal to 0 should be excluded from the output.
`include_diag`	A `logical` value indicating whether the diagonal (self-interactions) should be included in the output. This applies only to square matrices.
`include_lower`	A `logical` value indicating whether the lower triangular part of the `matrix` should be included in the output. This applies only to square matrices.

Value

A `data.frame` where each row represents the interaction between two nodes. If `weight = TRUE`, the `data.frame` includes a third column representing the weight of each interaction.

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a2_matrix_and_network_formats.html`.

Associated functions: net_to_mat

Examples

```
mat <- matrix(sample(1000, 50), 5, 10)
rownames(mat) <- paste0("Site", 1:5)
colnames(mat) <- paste0("Species", 1:10)

net <- mat_to_net(mat, weight = TRUE)
```

netclu_beckett — *Community structure detection in weighted bipartite networks via modularity optimization*

Description

This function takes a bipartite weighted graph and computes modules by applying Newman's modularity measure in a bipartite weighted version.

Usage

```
netclu_beckett(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  seed = NULL,
  forceLPA = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  algorithm_in_output = TRUE
)
```

Arguments

`net`	A `data.frame` representing a bipartite network with the first two columns representing undirected links between pairs of nodes, and the next column(s) representing the weights of the links.
`weight`	A `boolean` indicating whether weights should be considered if there are more than two columns (see Note).
`cut_weight`	A minimal weight value. If `weight` is TRUE, links with weights strictly lower than this value will not be considered (`0` by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`seed`	The seed for the random number generator (`NULL` for random by default).
`forceLPA`	A `boolean` indicating whether the even faster pure LPA-algorithm of Beckett should be used. DIRT-LPA (the default) is less likely to get trapped in a local minimum but is slightly slower. Defaults to `FALSE`.
`site_col`	The name or number of the column for site nodes (i.e., primary nodes).
`species_col`	The name or number of the column for species nodes (i.e., feature nodes).
`return_node_type`	A `character` indicating which types of nodes (`"site"`, `"species"`, or `"both"`) should be returned in the output (`"both"` by default).

`algorithm_in_output`
: A `boolean` indicating whether the original output of computeModules should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on the modularity optimization algorithm provided by Stephen Beckett (Beckett, 2016) as implemented in the bipartite package (computeModules).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, users can find the output of computeModules in the `algorithm` slot.

Note

Beckett's algorithm is designed to handle weighted bipartite networks. If `weight = FALSE`, a weight of 1 will be assigned to each pair of nodes. Ensure that the `site_col` and `species_col` arguments correctly identify the respective columns for site nodes (primary nodes) and species nodes (feature nodes). The type of nodes returned in the output can be selected using the `return_node_type` argument: `"both"` to include both node types, `"site"` to return only site nodes, or `"species"` to return only species nodes.

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)

References

Beckett SJ (2016) Improved community detection in weighted bipartite networks. *Royal Society Open Science* 3, 140536.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html`.

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

```
net <- data.frame(
  Site = c(rep("A", 2), rep("B", 3), rep("C", 2)),
  Species = c("a", "b", "a", "c", "d", "b", "d"),
  Weight = c(10, 100, 1, 20, 50, 10, 20))

com <- netclu_beckett(net)
```

---

netclu_greedy *Community structure detection via greedy optimization of modularity*

Description

This function finds communities in a (un)weighted undirected network via greedy optimization of modularity.

Usage

```
netclu_greedy(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  algorithm_in_output = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (0 by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`bipartite`	A `boolean` indicating if the network is bipartite (see Details).
`site_col`	The name or number for the column of site nodes (i.e. primary nodes).
`species_col`	The name or number for the column of species nodes (i.e. feature nodes).

`return_node_type`
: A `character` indicating what types of nodes (`site`, `species` or `both`) should be returned in the output (`return_node_type = "both"` by default).

`algorithm_in_output`
: A `boolean` indicating if the original output of cluster_fast_greedy should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on the fast greedy modularity optimization algorithm (Clauset et al., 2004) as implemented in the igraph package (cluster_fast_greedy).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: `character` containing the name of the algorithm
2. **args**: `list` of input arguments as provided by the user
3. **inputs**: `list` of characteristics of the clustering process
4. **algorithm**: `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`)
5. **clusters**: `data.frame` containing the clustering results

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find the output of cluster_fast_greedy.

Note

Although this algorithm was not primarily designed to deal with bipartite network, it is possible to consider the bipartite network as unipartite network (`bipartite = TRUE`).

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e. primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `both` to keep both types of nodes, `sites` to preserve only the sites nodes and `species` to preserve only the species nodes.

Author(s)

Maxime Lenormand (`<maxime.lenormand@inrae.fr>`)
Pierre Denelle (`<pierre.denelle@gmail.com>`)
Boris Leroy (`<leroy.boris@gmail.com>`)

References

Clauset A, Newman MEJ & Moore C (2004) Finding community structure in very large networks. *Phys. Rev. E* 70, 066111.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html.

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_greedy(net)

net_bip <- mat_to_net(comat, weight = TRUE)
clust2 <- netclu_greedy(net_bip, bipartite = TRUE)
```

---

netclu_infomap *Infomap community finding*

Description

This function finds communities in a (un)weighted (un)directed network based on the Infomap algorithm (https://github.com/mapequation/infomap).

Usage

```
netclu_infomap(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  seed = NULL,
  nbmod = 0,
  markovtime = 1,
  numtrials = 1,
  twolevel = FALSE,
  show_hierarchy = FALSE,
  directed = FALSE,
  bipartite_version = FALSE,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  version = "2.8.0",
  binpath = "tempdir",
  check_install = TRUE,
  path_temp = "infomap_temp",
  delete_temp = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (`0` by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`seed`	The seed for the random number generator (`NULL` for random by default).
`nbmod`	Penalize solutions the more they differ from this number (`0` by default for no preferred number of modules).
`markovtime`	Scales link flow to change the cost of moving between modules, higher values result in fewer modules (`1` by default).
`numtrials`	For the number of trials before picking up the best solution.
`twolevel`	A `boolean` indicating if the algorithm should optimize a two-level partition of the network (`FALSE` by default for multi-level).
`show_hierarchy`	A `boolean` specifying if the hierarchy of community should be identifiable in the outputs (`FALSE` by default).
`directed`	A `boolean` indicating if the network is directed (from column 1 to column 2).
`bipartite_version`	A `boolean` indicating if the bipartite version of Infomap should be used (see Note).
`bipartite`	A `boolean` indicating if the network is bipartite (see Note).
`site_col`	The name or number for the column of site nodes (i.e. primary nodes).
`species_col`	The name or number for the column of species nodes (i.e. feature nodes).
`return_node_type`	A `character` indicating what types of nodes (`"site"`, `"species"`, or `"both"`) should be returned in the output (`"both"` by default).
`version`	A `character` indicating the Infomap version to use.
`binpath`	A `character` indicating the path to the bin folder (see install_binaries and Details).
`check_install`	A `boolean` indicating if the function should check that the Infomap has been properly installed (see install_binaries and Details).
`path_temp`	A `character` indicating the path to the temporary folder (see Details).
`delete_temp`	A `boolean` indicating if the temporary folder should be removed (see Details).

Details

Infomap is a network clustering algorithm based on the Map equation proposed in Rosvall & Bergstrom (2008) that finds communities in (un)weighted and (un)directed networks.

This function is based on the C++ version of Infomap (`https://github.com/mapequation/infomap/releases`). This function needs binary files to run. They can be installed with install_binaries.

If you changed the default path to the `bin` folder while running install_binaries PLEASE MAKE SURE to set `binpath` accordingly.

If you did not use install_binaries to change the permissions and test the binary files PLEASE MAKE SURE to set `check_install` accordingly.

The C++ version of Infomap generates temporary folders and/or files that are stored in the `path_temp` folder ("infomap_temp" with a unique timestamp located in the bin folder in `binpath` by default). This temporary folder is removed by default (`delete_temp = TRUE`).

Several versions of Infomap are available in the package. See install_binaries for more details.

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects.
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, users can find the following elements:

- `cmd`: The command line used to run Infomap.
- `version`: The Infomap version.
- `web`: Infomap's GitHub repository.

Note

Infomap has been designed to deal with bipartite networks. To use this functionality, set the `bipartite_version` argument to TRUE in order to approximate a two-step random walker (see `https://www.mapequation.org/infomap/` for more information). Note that a bipartite network can also be considered as a unipartite network (`bipartite = TRUE`).

In both cases, do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e., primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `"both"` to keep both types of nodes, `"site"` to preserve only the site nodes, and `"species"` to preserve only the species nodes.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

References

Rosvall M & Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105, 1118-1123.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html.

Associated functions: netclu_greedy netclu_louvain netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_infomap(net)
```

---

netclu_labelprop *Finding communities based on propagating labels*

Description

This function finds communities in a (un)weighted undirected network based on propagating labels.

Usage

```
netclu_labelprop(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  seed = NULL,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  algorithm_in_output = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (`0` by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`seed`	The seed for the random number generator (`NULL` for random by default).
`bipartite`	A boolean indicating if the network is bipartite (see Details).
`site_col`	The name or number for the column of site nodes (i.e. primary nodes).
`species_col`	The name or number for the column of species nodes (i.e. feature nodes).
`return_node_type`	A `character` indicating what types of nodes (`"site"`, `"species"`, or `"both"`) should be returned in the output (`"both"` by default).
`algorithm_in_output`	A `boolean` indicating if the original output of cluster_label_prop should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on propagating labels (Raghavan et al., 2007) as implemented in the igraph package (cluster_label_prop).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find a "communities" object, output of cluster_label_prop.

Note

Although this algorithm was not primarily designed to deal with bipartite networks, it is possible to consider the bipartite network as a unipartite network (`bipartite = TRUE`).

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e., primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `"both"` to keep both types of nodes, `"site"` to preserve only the site nodes, and `"species"` to preserve only the species nodes.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

References

Raghavan UN, Albert R & Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 036106.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html.

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_labelprop(net)

net_bip <- mat_to_net(comat, weight = TRUE)
clust2 <- netclu_labelprop(net_bip, bipartite = TRUE)
```

`netclu_leadingeigen` *Finding communities based on the leading eigenvector of the community matrix*

Description

This function finds communities in a (un)weighted undirected network based on the leading eigenvector of the community matrix.

Usage

```
netclu_leadingeigen(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  algorithm_in_output = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (`0` by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`bipartite`	A `boolean` indicating if the network is bipartite (see Details).
`site_col`	The name or number for the column of site nodes (i.e., primary nodes).
`species_col`	The name or number for the column of species nodes (i.e., feature nodes).
`return_node_type`	A `character` indicating what types of nodes (`"site"`, `"species"`, or `"both"`) should be returned in the output (`"both"` by default).
`algorithm_in_output`	A `boolean` indicating if the original output of cluster_leading_eigen should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on the leading eigenvector of the community matrix (Newman, 2006) as implemented in the igraph package (cluster_leading_eigen).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find the output of cluster_leading_eigen.

Note

Although this algorithm was not primarily designed to deal with bipartite networks, it is possible to consider the bipartite network as a unipartite network (`bipartite = TRUE`).

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e., primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `"both"` to keep both types of nodes, `"site"` to preserve only the site nodes, and `"species"` to preserve only the species nodes.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

References

Newman MEJ (2006) Finding community structure in networks using the eigenvectors of matrices. *Physical Review E* 74, 036104.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html.

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_leadingeigen(net)

net_bip <- mat_to_net(comat, weight = TRUE)
clust2 <- netclu_leadingeigen(net_bip, bipartite = TRUE)
```

---

netclu_leiden *Finding communities using the Leiden algorithm*

Description

This function finds communities in a (un)weighted undirected network based on the Leiden algorithm of Traag, van Eck & Waltman.

Usage

```
netclu_leiden(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  seed = NULL,
  objective_function = "CPM",
  resolution_parameter = 1,
  beta = 0.01,
  n_iterations = 2,
  vertex_weights = NULL,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  algorithm_in_output = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (`0` by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`seed`	The random number generator seed (NULL for random by default).
`objective_function`	A string indicating the objective function to use, either the Constant Potts Model ("CPM") or "modularity" ("CPM" by default).
`resolution_parameter`	The resolution parameter to use. Higher resolutions lead to smaller communities, while lower resolutions lead to larger communities.
`beta`	A parameter affecting the randomness in the Leiden algorithm. This affects only the refinement step of the algorithm.
`n_iterations`	The number of iterations for the Leiden algorithm. Each iteration may further improve the partition.
`vertex_weights`	The vertex weights used in the Leiden algorithm. If not provided, they will be automatically determined based on the objective_function. Please see the details of this function to understand how to interpret the vertex weights.
`bipartite`	A `boolean` indicating if the network is bipartite (see Details).

`site_col`	The name or number for the column of site nodes (i.e., primary nodes).
`species_col`	The name or number for the column of species nodes (i.e., feature nodes).
`return_node_type`	A `character` indicating what types of nodes ("site", "species", or "both") should be returned in the output (`"both"` by default).
`algorithm_in_output`	A `boolean` indicating if the original output of cluster_leiden should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on the Leiden algorithm (Traag et al., 2019) as implemented in the igraph package (cluster_leiden).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find the output of cluster_leiden.

Note

Although this algorithm was not primarily designed to deal with bipartite networks, it is possible to consider the bipartite network as a unipartite network (`bipartite = TRUE`).

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e., primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `"both"` to keep both types of nodes, `"site"` to preserve only the site nodes, and `"species"` to preserve only the species nodes.

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)

References

Traag VA, Waltman L & Van Eck NJ (2019) From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 5233.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html.

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_leiden(net)

net_bip <- mat_to_net(comat, weight = TRUE)
clust2 <- netclu_leiden(net_bip, bipartite = TRUE)
```

`netclu_louvain` *Louvain community finding*

Description

This function finds communities in a (un)weighted undirected network based on the Louvain algorithm.

Usage

```
netclu_louvain(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  lang = "igraph",
  resolution = 1,
  seed = NULL,
  q = 0,
  c = 0.5,
  k = 1,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  binpath = "tempdir",
  check_install = TRUE,
  path_temp = "louvain_temp",
  delete_temp = TRUE,
  algorithm_in_output = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (`0` by default).
`index`	The name or number of the column to use as weight. By default, the third column name of `net` is used.
`lang`	A string indicating which version of Louvain should be used (`"igraph"` or `"cpp"`, see Details).
`resolution`	A resolution parameter to adjust the modularity (1 is chosen by default, see Details).
`seed`	The random number generator seed (only when `lang = "igraph"`, NULL for random by default).
`q`	The quality function used to compute the partition of the graph (modularity is chosen by default, see Details).
`c`	The parameter for the Owsinski-Zadrozny quality function (between 0 and 1, 0.5 is chosen by default).
`k`	The kappa_min value for the Shi-Malik quality function (it must be > 0, 1 is chosen by default).
`bipartite`	A `boolean` indicating if the network is bipartite (see Details).
`site_col`	The name or number for the column of site nodes (i.e., primary nodes).
`species_col`	The name or number for the column of species nodes (i.e., feature nodes).
`return_node_type`	A `character` indicating what types of nodes (`"site"`, `"species"`, or `"both"`) should be returned in the output (`"both"` by default).
`binpath`	A `character` indicating the path to the bin folder (see install_binaries and Details).
`check_install`	A `boolean` indicating if the function should check that Louvain has been properly installed (see install_binaries and Details).
`path_temp`	A `character` indicating the path to the temporary folder (see Details).
`delete_temp`	A `boolean` indicating if the temporary folder should be removed (see Details).
`algorithm_in_output`	A `boolean` indicating if the original output of cluster_louvain should be returned in the output (`TRUE` by default, see Value).

Details

Louvain is a network community detection algorithm proposed in (Blondel et al., 2008). This function offers two implementations of the Louvain algorithm (controlled by the `lang` parameter): the igraph implementation (cluster_louvain) and the C++ implementation (`https://sourceforge.net/projects/louvain/`, version 0.3).

The igraph implementation allows adjustment of the resolution parameter of the modularity function (`resolution` argument) used internally by the algorithm. Lower values typically yield fewer, larger clusters. The original definition of modularity is recovered when the resolution parameter is set to 1 (by default).

The C++ implementation provides several quality functions: `q = 0` for the classical Newman-Girvan criterion (Modularity), `q = 1` for the Zahn-Condorcet criterion, `q = 2` for the Owsinski-Zadrozny criterion (parameterized by `c`), `q = 3` for the Goldberg Density criterion, `q = 4` for the A-weighted Condorcet criterion, `q = 5` for the Deviation to Indetermination criterion, `q = 6` for the Deviation to Uniformity criterion, `q = 7` for the Profile Difference criterion, `q = 8` for the Shi-Malik criterion (parameterized by `k`), and `q = 9` for the Balanced Modularity criterion.

The C++ version is based on version 0.3 (`https://sourceforge.net/projects/louvain/`). Binary files are required to run it, and can be installed with install_binaries.

If you changed the default path to the `bin` folder while running install_binaries, PLEASE MAKE SURE to set `binpath` accordingly.

If you did not use install_binaries to change the permissions or test the binary files, PLEASE MAKE SURE to set `check_install` accordingly.

The C++ version generates temporary folders and/or files in the `path_temp` folder ("louvain_temp" with a unique timestamp located in the bin folder in `binpath` by default). This temporary folder is removed by default (`delete_temp = TRUE`).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find the output of cluster_louvain if `lang = "igraph"` and the following element if `lang = "cpp"`:

- `cmd`: The command line used to run Louvain.
- `version`: The Louvain version.
- `web`: The Louvain's website.

Note

Although this algorithm was not primarily designed to deal with bipartite networks, it is possible to consider the bipartite network as a unipartite network (`bipartite = TRUE`).

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e., primary nodes) and species nodes (i.e., feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `"both"` to keep both types of nodes, `"site"` to preserve only the site nodes, and `"species"` to preserve only the species nodes.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

References

Blondel VD, Guillaume JL, Lambiotte R & Mech ELJS (2008) Fast unfolding of communities in large networks. *J. Stat. Mech.* 10, P10008.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html.

Associated functions: netclu_infomap netclu_greedy netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_louvain(net, lang = "igraph")
```

`netclu_oslom` *OSLOM community finding*

Description

This function finds communities in a (un)weighted (un)directed network based on the OSLOM algorithm (http://oslom.org/, version 2.4).

Usage

```
netclu_oslom(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  seed = NULL,
  reassign = "no",
  r = 10,
  hr = 50,
  t = 0.1,
  cp = 0.5,
  directed = FALSE,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  binpath = "tempdir",
  check_install = TRUE,
  path_temp = "oslom_temp",
  delete_temp = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (0 by default).
`index`	Name or number of the column to use as weight. By default, the third column name of `net` is used.
`seed`	For the random number generator (NULL for random by default).
`reassign`	A `character` indicating if the nodes belonging to several community should be reassigned and what method should be used (see Note).
`r`	The number of runs for the first hierarchical level (10 by default).
`hr`	The number of runs for the higher hierarchical level (50 by default, 0 if you are not interested in hierarchies).
`t`	The p-value, the default value is 0.10. Increase this value if you want more modules.
`cp`	Kind of resolution parameter used to decide between taking some modules or their union (default value is 0.5; a bigger value leads to bigger clusters).
`directed`	A `boolean` indicating if the network is directed (from column 1 to column 2).

`bipartite`	A `boolean` indicating if the network is bipartite (see Details).
`site_col`	Name or number for the column of site nodes (i.e. primary nodes).
`species_col`	Name or number for the column of species nodes (i.e. feature nodes).
`return_node_type`	A `character` indicating what types of nodes (`site`, `species`, or `both`) should be returned in the output (`return_node_type = "both"` by default).
`binpath`	A `character` indicating the path to the bin folder (see install_binaries and Details).
`check_install`	A `boolean` indicating if the function should check that the OSLOM has been properly installed (see install_binaries and Details).
`path_temp`	A `character` indicating the path to the temporary folder (see Details).
`delete_temp`	A `boolean` indicating if the temporary folder should be removed (see Details).

Details

OSLOM is a network community detection algorithm proposed in Lancichinetti et al. (2011) that finds statistically significant (overlapping) communities in (un)weighted and (un)directed networks.

This function is based on the 2.4 C++ version of OSLOM (`http://www.oslom.org/software.htm`). This function needs files to run. They can be installed with install_binaries.

If you changed the default path to the `bin` folder while running install_binaries, PLEASE MAKE SURE to set `binpath` accordingly.

If you did not use install_binaries to change the permissions and test the binary files, PLEASE MAKE SURE to set `check_install` accordingly.

The C++ version of OSLOM generates temporary folders and/or files that are stored in the `path_temp` folder (folder "oslom_temp" with a unique timestamp located in the bin folder in `binpath` by default). This temporary folder is removed by default (`delete_temp = TRUE`).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, users can find the following elements:

- `cmd`: The command line used to run OSLOM.
- `version`: The OSLOM version.
- `web`: The OSLOM's web site.

Note

Although this algorithm was not primarily designed to deal with bipartite networks, it is possible to consider the bipartite network as unipartite network (`bipartite = TRUE`). Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e. primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `both` to keep both types of nodes, `sites` to preserve only the sites nodes, and `species` to preserve only the species nodes.

Since OSLOM potentially returns overlapping communities, we propose two methods to reassign the 'overlapping' nodes: randomly (`reassign = "random"`) or based on the closest candidate community (`reassign = "simil"`) (only for weighted networks, in this case the closest candidate community is determined with the average similarity). By default, `reassign = "no"` and all the information will be provided. The number of partitions will depend on the number of overlapping modules (up to three). The suffix `_semel`, `_bis`, and `_ter` are added to the column names. The first partition (`_semel`) assigns a module to each node. A value of `NA` in the second (`_bis`) and third (`_ter`) columns indicates that no overlapping module was found for this node (i.e. non-overlapping nodes).

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)

References

Lancichinetti A, Radicchi F, Ramasco JJ & Fortunato S (2011) Finding statistically significant communities in networks. *PLOS ONE* 6, e18961.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html`.

Associated functions: netclu_greedy netclu_infomap netclu_louvain

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_oslom(net)
```

netclu_walktrap *Community structure detection via short random walks*

Description

This function finds communities in a (un)weighted undirected network via short random walks.

Usage

```
netclu_walktrap(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  steps = 4,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  algorithm_in_output = TRUE
)
```

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (0 by default).
`index`	Name or number of the column to use as weight. By default, the third column name of `net` is used.
`steps`	The length of the random walks to perform.
`bipartite`	A `boolean` indicating if the network is bipartite (see Details).
`site_col`	Name or number for the column of site nodes (i.e. primary nodes).
`species_col`	Name or number for the column of species nodes (i.e. feature nodes).
`return_node_type`	A `character` indicating what types of nodes (`site`, `species`, or `both`) should be returned in the output (`return_node_type = "both"` by default).
`algorithm_in_output`	A `boolean` indicating if the original output of cluster_walktrap should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on random walks (Pons & Latapy, 2005) as implemented in the igraph package (cluster_walktrap).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

In the `algorithm` slot, if `algorithm_in_output = TRUE`, users can find the output of cluster_walktrap.

Note

Although this algorithm was not primarily designed to deal with bipartite networks, it is possible to consider the bipartite network as unipartite network (`bipartite = TRUE`).

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e. primary nodes) and species nodes (i.e. feature nodes) using the arguments `site_col` and `species_col`. The type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `both` to keep both types of nodes, `sites` to preserve only the site nodes, and `species` to preserve only the species nodes.

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)

References

Pons P & Latapy M (2005) Computing Communities in Large Networks Using Random Walks. In Yolum I, Güngör T, Gürgen F, Özturan C (eds.), *Computer and Information Sciences - ISCIS 2005*, Lecture Notes in Computer Science, 284-293.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_3_network_clustering.html`.

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

```
comat <- matrix(sample(1000, 50), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

net <- similarity(comat, metric = "Simpson")
com <- netclu_walktrap(net)

net_bip <- mat_to_net(comat, weight = TRUE)
clust2 <- netclu_walktrap(net_bip, bipartite = TRUE)
```

net_to_mat *Create a contingency table from a data.frame*

Description

This function generates a contingency table from a two- or three-column `data.frame`, where each row represents the interaction between two nodes (e.g., site and species) and an optional third column indicates the weight of the interaction (if `weight = TRUE`).

Usage

```
net_to_mat(
  net,
  weight = FALSE,
  squared = FALSE,
  symmetrical = FALSE,
  missing_value = 0
)
```

Arguments

`net`	A two- or three-column `data.frame` where each row represents the interaction between two nodes (e.g., site and species), with an optional third column indicating the weight of the interaction.
`weight`	A `logical` value indicating whether the weight column should be considered.
`squared`	A `logical` value indicating whether the output matrix should be square (i.e., containing the same nodes in rows and columns).
`symmetrical`	A `logical` value indicating whether the resulting matrix should be symmetrical. This applies only if `squared = TRUE`. Note that different weights associated with opposite pairs already present in `net` will be preserved.
`missing_value`	The value to assign to pairs of nodes not present in `net`. Defaults to `0`.

Value

A `matrix` with the first nodes (from the first column of `net`) as rows and the second nodes (from the second column of `net`) as columns. If `squared = TRUE`, the rows and columns will have the same number of elements, corresponding to the unique union of objects in the first and second columns of `net`. If `squared = TRUE` and `symmetrical = TRUE`, the matrix will be forced to be symmetrical based on the upper triangular part of the matrix.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a2_matrix_and_network_formats.html.

Associated functions: mat_to_net

Examples

```
net <- data.frame(
  Site = c(rep("A", 2), rep("B", 3), rep("C", 2)),
  Species = c("a", "b", "a", "c", "d", "b", "d"),
  Weight = c(10, 100, 1, 20, 50, 10, 20)
)

mat <- net_to_mat(net, weight = TRUE)
```

`nhclu_affprop` *Non-hierarchical clustering: Affinity Propagation*

Description

This function performs non-hierarchical clustering using the Affinity Propagation algorithm.

Usage

```
nhclu_affprop(
  similarity,
  index = names(similarity)[3],
  seed = NULL,
  p = NA,
  q = NA,
  maxits = 1000,
  convits = 100,
  lam = 0.9,
  details = FALSE,
  nonoise = FALSE,
  K = NULL,
  prc = NULL,
  bimaxit = NULL,
  exact = NULL,
  algorithm_in_output = TRUE
)
```

Arguments

`similarity`	The output object from `similarity()` or `dissimilarity_to_similarity()`, or a `dist` object. If a `data.frame` is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the similarity indices.
`index`	The name or number of the similarity column to use. By default, the third column name of `similarity` is used.
`seed`	The seed for the random number generator used when `nonoise = FALSE`.
`p`	Input preference, which can be a vector specifying individual preferences for each data point. If scalar, the same value is used for all data points. If `NA`, exemplar preferences are initialized based on the distribution of non-Inf values in the similarity matrix, controlled by `q`.
`q`	If `p = NA`, exemplar preferences are initialized according to the distribution of non-Inf values in the similarity matrix. By default, the median is used. A value between 0 and 1 specifies the sample quantile, where `q = 0.5` results in the median.
`maxits`	The maximum number of iterations to execute.
`convits`	The algorithm terminates if the exemplars do not change for `convits` iterations.
`lam`	The damping factor, a value in the range [0.5, 1). Higher values correspond to heavier damping, which may help prevent oscillations.
`details`	If `TRUE`, detailed information about the algorithm's progress is stored in the output object.
`nonoise`	If `TRUE`, disables the addition of a small amount of noise to the similarity object, which prevents degenerate cases.
`K`	The desired number of clusters. If not `NULL`, the function apclusterK is called.
`prc`	A parameter needed when `K` is not `NULL`. The algorithm stops if the number of clusters deviates by less than `prc` percent from the desired value `K`. Set to 0 to enforce exactly `K` clusters.
`bimaxit`	A parameter needed when `K` is not `NULL`. Specifies the maximum number of bisection steps to perform. No warning is issued if the number of clusters remains outside the desired range.
`exact`	A flag indicating whether to compute the initial preference range exactly.
`algorithm_in_output`	A `boolean` indicating whether to include the original output of apcluster in the result. Defaults to `TRUE`.

Details

This function is based on the apcluster package (apcluster).

Value

A `list` of class `bioregion.clusters` with five slots:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` describing the characteristics of the clustering process.
4. **algorithm**: A `list` of objects associated with the clustering procedure, such as original cluster objects (if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, the `algorithm` slot includes the output of apcluster.

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

References

Frey B & Dueck D (2007) Clustering by Passing Messages Between Data Points. *Science* 315, 972-976.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_2_non_hierarchical_clustering.html`.

Associated functions: nhclu_clara nhclu_clarans nhclu_dbscan nhclu_kmeans nhclu_affprop

Examples

```
comat_1 <- matrix(sample(0:1000, size = 10*12, replace = TRUE,
prob = 1/1:1001), 10, 12)
rownames(comat_1) <- paste0("Site", 1:10)
colnames(comat_1) <- paste0("Species", 1:12)
comat_1 <- cbind(comat_1,
                 matrix(0, 10, 8,
                        dimnames = list(paste0("Site", 1:10),
                                        paste0("Species", 13:20))))

comat_2 <- matrix(sample(0:1000,
                         size = 10*12,
                         replace = TRUE,
                         prob = 1/1:1001),
                  10, 12)
```

```
rownames(comat_2) <- paste0("Site", 11:20)
colnames(comat_2) <- paste0("Species", 9:20)
comat_2 <- cbind(matrix(0, 10, 8,
                        dimnames = list(paste0("Site", 11:20),
                                        paste0("Species", 1:8))),
                 comat_2)

comat <- rbind(comat_1, comat_2)

dissim <- dissimilarity(comat, metric = "Simpson")
sim <- dissimilarity_to_similarity(dissim)

clust1 <- nhclu_affprop(sim)

clust2 <- nhclu_affprop(sim, q = 1)

# Fixed number of clusters
clust3 <- nhclu_affprop(sim, K = 2, prc = 10, bimaxit = 20, exact = FALSE)
```

nhclu_clara *Non-hierarchical clustering: CLARA*

Description

This function performs non-hierarchical clustering based on dissimilarity using partitioning around medoids, implemented via the Clustering Large Applications (CLARA) algorithm.

Usage

```
nhclu_clara(
  dissimilarity,
  index = names(dissimilarity)[3],
  seed = NULL,
  n_clust = c(1, 2, 3),
  maxiter = 0,
  initializer = "LAB",
  fasttol = 1,
  numsamples = 5,
  sampling = 0.25,
  independent = FALSE,
  algorithm_in_output = TRUE
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.

`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`seed`	A value for the random number generator (set to `NULL` for random initialization by default).
`n_clust`	An `integer` vector or a single `integer` specifying the desired number(s) of clusters.
`maxiter`	An `integer` defining the maximum number of iterations.
`initializer`	A `character` string, either `"BUILD"` (used in the classic PAM algorithm) or `"LAB"` (Linear Approximate BUILD).
`fasttol`	A positive `numeric` value defining the tolerance for fast swapping behavior. Defaults to 1.
`numsamples`	A positive `integer` specifying the number of samples to draw.
`sampling`	A positive `numeric` value defining the sampling rate.
`independent`	A `boolean` indicating whether the previous medoids are excluded in the next sample. Defaults to `FALSE`.
`algorithm_in_output`	A `boolean` indicating whether the original output of fastclara should be included in the output. Defaults to `TRUE` (see Value).

Details

Based on fastkmedoids package (fastclara).

Value

A `list` of class `bioregion.clusters` with five components:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, the `algorithm` slot includes the output of fastclara.

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

References

Schubert E & Rousseeuw PJ (2019) Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. *Similarity Search and Applications* 11807, 171-187.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_2_non_hierarchical_clustering.html.

Associated functions: nhclu_clarans nhclu_dbscan nhclu_kmeans nhclu_pam nhclu_affprop

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "all")

#clust <- nhclu_clara(dissim, index = "Simpson", n_clust = 5)
```

---

nhclu_clarans *Non-hierarchical clustering: CLARANS*

Description

This function performs non-hierarchical clustering based on dissimilarity using partitioning around medoids, implemented via the Clustering Large Applications based on RANdomized Search (CLARANS) algorithm.

Usage

```
nhclu_clarans(
  dissimilarity,
  index = names(dissimilarity)[3],
  seed = NULL,
  n_clust = c(1, 2, 3),
  numlocal = 2,
  maxneighbor = 0.025,
  algorithm_in_output = TRUE
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.

`seed`	A value for the random number generator (`NULL` for random initialization by default).
`n_clust`	An `integer` vector or a single `integer` specifying the desired number(s) of clusters.
`numlocal`	An `integer` defining the number of local searches to perform.
`maxneighbor`	A positive `numeric` value defining the maximum number of neighbors to consider for each local search.
`algorithm_in_output`	A `boolean` indicating whether the original output of fastclarans should be included in the output. Defaults to `TRUE` (see Value).

Details

Based on fastkmedoids package (fastclarans).

Value

A `list` of class `bioregion.clusters` with five components:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, the `algorithm` slot includes the output of fastclarans.

Author(s)

Pierre Denelle (`<pierre.denelle@gmail.com>`)
Boris Leroy (`<leroy.boris@gmail.com>`)
Maxime Lenormand (`<maxime.lenormand@inrae.fr>`)

References

Schubert E & Rousseeuw PJ (2019) Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. *Similarity Search and Applications* 11807, 171-187.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_2_non_hierarchical_clustering.html`.

Associated functions: nhclu_clara nhclu_dbscan nhclu_kmeans nhclu_pam nhclu_affprop

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "all")

#clust <- nhclu_clarans(dissim, index = "Simpson", n_clust = 5)
```

---

nhclu_dbscan *Non-hierarchical clustering: DBSCAN*

Description

This function performs non-hierarchical clustering based on dissimilarity using the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm.

Usage

```
nhclu_dbscan(
  dissimilarity,
  index = names(dissimilarity)[3],
  minPts = NULL,
  eps = NULL,
  plot = TRUE,
  algorithm_in_output = TRUE,
  ...
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`minPts`	A `numeric` vector or a single `numeric` value specifying the `minPts` argument of `dbscan::dbscan()`. `minPts` is the minimum number of points to form a dense region. By default, it is set to the natural logarithm of the number of sites in `dissimilarity`. See Details for guidance on choosing this parameter.
`eps`	A `numeric` vector or a single `numeric` value specifying the `eps` argument of `dbscan::dbscan()`. `eps` specifies how similar points should be to each other to be considered part of a cluster. See Details for guidance on choosing this parameter.

`plot`	A `boolean` indicating whether the k-nearest neighbor distance plot should be displayed.
`algorithm_in_output`	A `boolean` indicating whether the original output of dbscan::dbscan should be included in the output. Defaults to `TRUE` (see Value).
`...`	Additional arguments to be passed to `dbscan()` (see dbscan::dbscan).

Details

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm clusters points based on the density of neighbors around each data point. It requires two main arguments: `minPts`, the minimum number of points to identify a core, and `eps`, the radius used to find neighbors.

Choosing minPts: This determines how many points are necessary to form a cluster. For example, what is the minimum number of sites expected in a bioregion? Choose a value sufficiently large for your dataset and expectations.

Choosing eps: This determines how similar sites should be to form a cluster. If `eps` is too small, most points will be considered too distinct and marked as noise. If `eps` is too large, clusters may merge. The value of `eps` depends on `minPts`. It is recommended to choose `eps` by identifying a knee in the k-nearest neighbor distance plot.

By default, the function attempts to find a knee in this curve automatically, but the result is uncertain. Users should inspect the graph and modify `eps` accordingly. To explore `eps` values, run the function initially without defining `eps`, review the recommendations, and adjust as needed based on clustering results.

Value

A `list` of class `bioregion.clusters` with five components:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, the `algorithm` slot includes the output of dbscan::dbscan.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

References

Hahsler M, Piekenbrock M & Doran D (2019) Dbscan: Fast density-based clustering with R. *Journal of Statistical Software*, 91(1), 1–30.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a4_2_non_hierarchical_clustering.html.

Associated functions: nhclu_clara nhclu_clarans nhclu_kmeans nhclu_pam nhclu_affprop

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "all")

clust1 <- nhclu_dbscan(dissim, index = "Simpson")
clust2 <- nhclu_dbscan(dissim, index = "Simpson", eps = 0.2)
clust3 <- nhclu_dbscan(dissim, index = "Simpson", minPts = c(5, 10, 15, 20),
     eps = c(.1, .15, .2, .25, .3))
```

nhclu_kmeans *Non-hierarchical clustering: K-means analysis*

Description

This function performs non-hierarchical clustering based on dissimilarity using a k-means analysis.

Usage

```
nhclu_kmeans(
  dissimilarity,
  index = names(dissimilarity)[3],
  seed = NULL,
  n_clust = c(1, 2, 3),
  iter_max = 10,
  nstart = 10,
  algorithm = "Hartigan-Wong",
  algorithm_in_output = TRUE
)
```

Arguments

dissimilarity	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
index	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.

`seed`	A value for the random number generator (`NULL` for random by default).
`n_clust`	An `integer` vector or a single `integer` value specifying the requested number(s) of clusters.
`iter_max`	An `integer` specifying the maximum number of iterations for the k-means method (see kmeans).
`nstart`	An `integer` specifying how many random sets of `n_clust` should be selected as starting points for the k-means analysis (see kmeans).
`algorithm`	A `character` specifying the algorithm to use for k-means (see kmeans). Available options are Hartigan-Wong, Lloyd, Forgy, and MacQueen.
`algorithm_in_output`	A `boolean` indicating whether the original output of kmeans should be included in the output. Defaults to `TRUE` (see Value).

Details

This method partitions data into k groups such that the sum of squares of Euclidean distances from points to the assigned cluster centers is minimized. K-means cannot be applied directly to dissimilarity or beta-diversity metrics because these distances are not Euclidean. Therefore, it first requires transforming the dissimilarity matrix using Principal Coordinate Analysis (PCoA) with pcoa, and then applying k-means to the coordinates of points in the PCoA.

Because this additional transformation alters the initial dissimilarity matrix, the partitioning around medoids method (nhclu_pam) is preferred.

Value

A `list` of class `bioregion.clusters` with five components:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, the `algorithm` slot includes the output of kmeans.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_2_non_hierarchical_clustering.html`.

Associated functions: nhclu_clara nhclu_clarans nhclu_dbscan nhclu_pam nhclu_affprop

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

comnet <- mat_to_net(comat)

dissim <- dissimilarity(comat, metric = "all")

clust <- nhclu_kmeans(dissim, n_clust = 2:10, index = "Simpson")
```

`nhclu_pam` *Non-hierarchical clustering: Partitioning Around Medoids*

Description

This function performs non-hierarchical clustering based on dissimilarity using partitioning around medoids (PAM).

Usage

```
nhclu_pam(
  dissimilarity,
  index = names(dissimilarity)[3],
  seed = NULL,
  n_clust = c(1, 2, 3),
  variant = "faster",
  nstart = 1,
  cluster_only = FALSE,
  algorithm_in_output = TRUE,
  ...
)
```

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`seed`	A value for the random number generator (`NULL` for random by default).
`n_clust`	An `integer` vector or a single `integer` value specifying the requested number(s) of clusters.

`variant`	A `character` string specifying the PAM variant to use. Defaults to `faster`. Available options are `original`, `o_1`, `o_2`, `f_3`, `f_4`, `f_5`, or `faster`. See pam for more details.
`nstart`	An `integer` specifying the number of random starts for the PAM algorithm. Defaults to 1 (for the `faster` variant).
`cluster_only`	A `boolean` specifying whether only the clustering results should be returned from the pam function. Setting this to `TRUE` makes the function more efficient.
`algorithm_in_output`	A `boolean` indicating whether the original output of pam should be included in the result. Defaults to `TRUE` (see Value).
`...`	Additional arguments to pass to `pam()` (see pam).

Details

This method partitions the data into the chosen number of clusters based on the input dissimilarity matrix. It is more robust than k-means because it minimizes the sum of dissimilarities between cluster centers (medoids) and points assigned to the cluster. In contrast, k-means minimizes the sum of squared Euclidean distances, which makes it unsuitable for dissimilarity matrices that are not based on Euclidean distances.

Value

A `list` of class `bioregion.clusters` with five components:

1. **name**: A `character` string containing the name of the algorithm.
2. **args**: A `list` of input arguments as provided by the user.
3. **inputs**: A `list` of characteristics of the clustering process.
4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

If `algorithm_in_output = TRUE`, the `algorithm` slot includes the output of pam.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)

References

Kaufman L & Rousseeuw PJ (2009) Finding groups in data: An introduction to cluster analysis. In & Sons. JW (ed.), Finding groups in data: An introduction to cluster analysis.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a4_2_non_hierarchical_clustering.html`.

Associated functions: nhclu_clara nhclu_clarans nhclu_dbscan nhclu_kmeans nhclu_affprop

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

comnet <- mat_to_net(comat)
dissim <- dissimilarity(comat, metric = "all")

clust <- nhclu_pam(dissim, n_clust = 2:15, index = "Simpson")
```

---

similarity *Compute similarity metrics between sites based on species composition*

Description

This function generates a `data.frame` where each row provides one or several similarity metrics between pairs of sites, based on a co-occurrence `matrix` with sites as rows and species as columns.

Usage

```
similarity(comat, metric = "Simpson", formula = NULL, method = "prodmat")
```

Arguments

`comat`	A co-occurrence `matrix` with sites as rows and species as columns.
`metric`	A `character` vector or a single `character` string specifying the metrics to compute (see Details). Available options are `"abc"`, `"ABC"`, `"Jaccard"`, `"Jaccardturn"`, `"Sorensen"`, `"Simpson"`, `"Bray"`, `"Brayturn"`, and `"Euclidean"`. If `"all"` is specified, all metrics will be calculated. Can be set to `NULL` if `formula` is used.
`formula`	A `character` vector or a single `character` string specifying custom formula(s) based on the `a`, `b`, `c`, `A`, `B`, and `C` quantities (see Details). The default is `NULL`.
`method`	A `character` string specifying the method to compute `abc` (see Details). The default is `"prodmat"`, which is more efficient but memory-intensive. Alternatively, `"loops"` is less memory-intensive but slower.

Details

With `a` the number of species shared by a pair of sites, `b` species only present in the first site and `c` species only present in the second site.

Jaccard = 1 - (b + c) / (a + b + c)

Jaccardturn = 1 - 2min(b, c) / (a + 2min(b, c)) (Baselga, 2012)

Sorensen = 1 - (b + c) / (2a + b + c)

Simpson = 1 - min(b, c) / (a + min(b, c))

If abundances data are available, Bray-Curtis and its turnover component can also be computed with the following equation:

Bray = 1 - (B + C) / (2A + B + C)

Brayturn = 1 - min(B, C) / (A + min(B, C)) (Baselga, 2013)

with `A` the sum of the lesser values for common species shared by a pair of sites. `B` and `C` are the total number of specimens counted at both sites minus `A`.

`formula` can be used to compute customized metrics with the terms `a`, `b`, `c`, `A`, `B`, and `C`. For example `formula = c("1 - pmin(b,c) / (a + pmin(b,c))", "1 - (B + C) / (2*A + B + C)")` will compute the Simpson and Bray-Curtis similarity metrics, respectively. Note that `pmin` is used in the Simpson formula because `a`, `b`, `c`, `A`, `B` and `C` are `numeric` vectors.

Euclidean computes the Euclidean similarity between each pair of site following this equation:

Euclidean = 1 / (1 + d_ij)

Where d_ij is the Euclidean distance between site i and site j in terms of species composition.

Value

A `data.frame` with the additional class `bioregion.pairwise.metric`, containing one or several similarity metrics between pairs of sites. The first two columns represent the pairs of sites. There is one column per similarity metric provided in `metric` and `formula`, except for the `abc` and `ABC` metrics, which are stored in three separate columns (one for each letter).

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>)
Pierre Denelle (<`pierre.denelle@gmail.com`>)
Boris Leroy (<`leroy.boris@gmail.com`>)

References

Baselga A (2012) The Relationship between Species Replacement, Dissimilarity Derived from Nestedness, and Nestedness. *Global Ecology and Biogeography* 21, 1223–1232.

Baselga A (2013) Separating the two components of abundance-based dissimilarity: balanced changes in abundance vs. abundance gradients. *Methods in Ecology and Evolution* 4, 552–557.

See Also

For more details illustrated with a practical example, see the vignette: `https://biorgeo.github.io/bioregion/articles/a3_pairwise_metrics.html`.

Associated functions: dissimilarity similarity_to_dissimilarity

Examples

```
comat <- matrix(sample(0:1000, size = 50, replace = TRUE,
prob = 1 / 1:1001), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)
```

```
sim <- similarity(comat, metric = c("abc", "ABC", "Simpson", "Brayturn"))

sim <- similarity(comat, metric = "all",
formula = "1 - (b + c) / (a + b + c)")
```

---

similarity_to_dissimilarity

Convert similarity metrics to dissimilarity metrics

Description

This function converts a `data.frame` of similarity metrics between sites into dissimilarity metrics (beta diversity).

Usage

```
similarity_to_dissimilarity(similarity, include_formula = TRUE)
```

Arguments

`similarity`	The output object from `similarity()` or `dissimilarity_to_similarity()`.
`include_formula`	A `boolean` indicating whether metrics based on custom formula(s) should also be converted (see Details). The default is `TRUE`.

Value

A `data.frame` with additional class `bioregion.pairwise.metric`, providing dissimilarity metric(s) between each pair of sites based on a similarity object.

Note

The behavior of this function changes depending on column names. Columns `Site1` and `Site2` are copied identically. If there are columns called `a`, `b`, `c`, `A`, `B`, `C` they will also be copied identically. If there are columns based on your own formula (argument `formula` in `similarity()`) or not in the original list of similarity metrics (argument `metrics` in `similarity()`) and if the argument `include_formula` is set to `FALSE`, they will also be copied identically. Otherwise there are going to be converted like they other columns (default behavior).

If a column is called `Euclidean`, its distance will be calculated based on the following formula:

Euclidean distance = (1 - Euclidean similarity) / Euclidean similarity

Otherwise, all other columns will be transformed into dissimilarity with the following formula:

dissimilarity = 1 - similarity

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Boris Leroy (<leroy.boris@gmail.com>)
Pierre Denelle (<pierre.denelle@gmail.com>)

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a3_pairwise_metrics.html.

Associated functions: dissimilarity similarity_to_dissimilarity

Examples

```
comat <- matrix(sample(0:1000, size = 50, replace = TRUE,
prob = 1 / 1:1001), 5, 10)
rownames(comat) <- paste0("Site", 1:5)
colnames(comat) <- paste0("Species", 1:10)

simil <- similarity(comat, metric = "all")
simil

dissimilarity <- similarity_to_dissimilarity(simil)
dissimilarity
```

`site_species_metrics` *Calculate contribution metrics of sites and species*

Description

This function calculates metrics to assess the contribution of a given species or site to its bioregion.

Usage

```
site_species_metrics(
  bioregionalization,
  comat,
  indices = c("rho"),
  net = NULL,
  site_col = 1,
  species_col = 2
)
```

Arguments

`bioregionalization`	A `bioregion.clusters` object.
`comat`	A co-occurrence `matrix` with sites as rows and species as columns.
`indices`	A `character` specifying the contribution metric to compute. Available options are `rho`, `affinity`, `fidelity`, `indicator_value` and `Cz`.
`net`	`NULL` by default. Required for `Cz` indices. A `data.frame` where each row represents an interaction between two nodes and an optional third column indicating the interaction's weight.
`site_col`	A number indicating the position of the column containing the sites in `net`. 1 by default.
`species_col`	A number indicating the position of the column containing the species in `net`. 2 by default.

Details

The ρ metric is derived from Lenormand et al. (2019) with the following formula:

$$\rho_{ij} = \frac{n_{ij} - \frac{n_i n_j}{n}}{\sqrt{\left(\frac{n-n_j}{n-1}\right)\left(1-\frac{n_j}{n}\right)\frac{n_i n_j}{n}}}$$

where n is the number of sites, n_i is the number of sites in which species i is present, n_j is the number of sites in bioregion j, and n_{ij} is the number of occurrences of species i in sites of bioregion j.

Affinity A, fidelity F, and individual contributions $IndVal$ describe how species are linked to their bioregions. These metrics are described in Bernardo-Madrid et al. (2019):

- Affinity of species to their region: $A_i = \frac{R_i}{Z}$, where R_i is the occurrence/range size of species i in its associated bioregion, and Z is the total size (number of sites) of the bioregion. High affinity indicates that the species occupies most sites in its bioregion.
- Fidelity of species to their region: $F_i = \frac{R_i}{D_i}$, where R_i is the occurrence/range size of species i in its bioregion, and D_i is its total range size. High fidelity indicates that the species is not present in other regions.
- Indicator Value of species: $IndVal = F_i \cdot A_i$.

`Cz` metrics are derived from Guimerà & Amaral (2005):

- Participation coefficient: $C_i = 1 - \sum_{s=1}^{N_M} \left(\frac{k_{is}}{k_i}\right)^2$, where k_{is} is the number of links of node i to nodes in bioregion s, and k_i is the total degree of node i. A high value means links are uniformly distributed; a low value means links are within the node's bioregion.
- Within-bioregion degree z-score: $z_i = \frac{k_i - \overline{k_{si}}}{\sigma_{k_{si}}}$, where k_i is the number of links of node i to nodes in its bioregion s_i, $\overline{k_{si}}$ is the average degree of nodes in s_i, and $\sigma_{k_{si}}$ is the standard deviation of degrees in s_i.

Value

A `data.frame` with columns `Bioregion`, `Species`, and the desired summary statistics, or a list of `data.frames` if `Cz` and other indices are selected.

Author(s)

Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)
Maxime Lenormand (<maxime.lenormand@inrae.fr>)

References

Bernardo-Madrid R, Calatayud J, González-Suárez M, Rosvall M, Lucas P, Antonelli A & Revilla E (2019) Human activity is altering the world's zoogeographical regions. *Ecology Letters* 22, 1297–1305.

Guimerà R & Amaral LAN (2005) Functional cartography of complex metabolic networks. *Nature* 433, 895–900.

Lenormand M, Papuga G, Argagnon O, Soubeyrand M, Alleaume S & Luque S (2019) Biogeographical network analysis of plant species distribution in the Mediterranean region. *Ecology and Evolution* 9, 237–250.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_3_summary_metrics.html.

Associated functions: bioregion_metrics bioregionalization_metrics

Examples

```
comat <- matrix(sample(0:1000, size = 500, replace = TRUE, prob = 1/1:1001),
                20, 25)
rownames(comat) <- paste0("Site",1:20)
colnames(comat) <- paste0("Species",1:25)

dissim <- dissimilarity(comat, metric = "Simpson")
clust1 <- nhclu_kmeans(dissim, n_clust = 3, index = "Simpson")

net <- similarity(comat, metric = "Simpson")
com <- netclu_greedy(net)

site_species_metrics(bioregionalization = clust1, comat = comat,
indices = "rho")

# Contribution metrics
site_species_metrics(bioregionalization = com, comat = comat,
indices = c("rho", "affinity", "fidelity", "indicator_value"))

# Cz indices
net_bip <- mat_to_net(comat, weight = TRUE)
clust_bip <- netclu_greedy(net_bip, bipartite = TRUE)
site_species_metrics(bioregionalization = clust_bip, comat = comat,
net = net_bip, indices = "Cz")
```

`site_species_subset` *Extract a subset of sites or species from a* `bioregion.clusters` *object*

Description

This function extracts a subset of nodes based on their type (`"site"` or `"species"`) from a `bioregion.clusters` object, which contains both types of nodes (sites and species).

Usage

```
site_species_subset(clusters, node_type = "site")
```

Arguments

`clusters`	An object of class `bioregion.clusters`.
`node_type`	A `character` string indicating the type of nodes to extract. Possible values are `"site"` or `"species"`. The default is `"site"`.

Value

An object of class `bioregion.clusters` containing only the specified node type (sites or species).

Note

Network clustering functions (prefixed with `netclu_`) may return both types of nodes (sites and species) when applied to bipartite networks (using the `bipartite` argument). In such cases, the type of nodes included in the output can be specified with the `return_node_type` argument. This function allows you to extract a particular type of nodes (sites or species) from the output and adjust the `return_node_type` attribute accordingly.

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>)
Pierre Denelle (<pierre.denelle@gmail.com>)
Boris Leroy (<leroy.boris@gmail.com>)

Examples

```
net <- data.frame(
  Site = c(rep("A", 2), rep("B", 3), rep("C", 2)),
  Species = c("a", "b", "a", "c", "d", "b", "d"),
  Weight = c(10, 100, 1, 20, 50, 10, 20)
)

clusters <- netclu_louvain(net, lang = "igraph", bipartite = TRUE)

clusters_sites <- site_species_subset(clusters, node_type = "site")
```

`vegedf` *Spatial distribution of Mediterranean vegetation (data.frame)*

Description

A dataset containing the abundance of 3,697 species in 715 sites.

Usage

```
vegedf
```

Format

A `data.frame` with 460,878 rows and 3 columns:

Site Unique site identifier (corresponding to the field ID of vegesp)

Species Unique species identifier

Abundance Species abundance

Source

doi:10.1002/ece3.4718

`vegemat` *Spatial distribution of Mediterranean vegetation (co-occurrence matrix)*

Description

A dataset containing the abundance of each of the 3,697 species in each of the 715 sites.

Usage

```
vegemat
```

Format

A co-occurrence `matrix` with sites as rows and species as columns. Each element of the matrix represents the abundance of the species in the site.

Source

doi:10.1002/ece3.4718

`vegesf` *Spatial distribution of Mediterranean vegetation (spatial grid)*

Description

A dataset containing the geometry of the 715 sites.

Usage

```
vegesf
```

Format

A

ID Unique site identifier

geometry Geometry of the site

Source

doi:10.1002/ece3.4718

Index