Package: bioregion (via r-universe)

September 13, 2024

Type Package

Title Comparison of Bioregionalisation Methods

Version 1.1.1

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 (Lenormand 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, bipartite, cluster, data.table, dbscan, dynamicTreeCut, fastcluster, fastkmedoids, ggplot2, grDevices, igraph, mathjaxr, Matrix, 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

Roxygen list(markdown = TRUE)

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

RemoteUrl https://github.com/biorgeo/bioregion

RemoteRef HEAD

RemoteSha 5a55165e662346d84636740b7a0d0d37f7c089f8

Contents

compare_partitions *Compare cluster memberships among multiple partitions*

Description

This function aims at computing pairwise comparisons for several partitions, usually on outputs from `netclu_`, `hclu_` or `nhclu_` functions. It also provides the confusion matrix from pairwise comparisons, so that the user can compute additional comparison metrics.

Usage

```
compare_partitions(
  cluster_object,
  sample_items = NULL,
  indices = c("rand", "jaccard"),
  cor_frequency = FALSE,
  store_pairwise_membership = TRUE,
  store_confusion_matrix = TRUE
)
```

Arguments

`cluster_object`	a `bioregion.clusters` object or a `data.frame` or a list of `data.frame` containing multiple partitions. At least two partitions are required. If a list of `data.frame` is provided, they should all have the same number of rows (i.e., same items in the clustering for all partitions).
`sample_items`	`NULL` or a positive integer. Reduce the number of items to be used in the comparison of partitions. Useful if the number of items is high and pairwise comparisons cannot be computed. Suggested values 5000 or 10000 computation
`indices`	`NULL` or `character`. Indices to compute for the pairwise comparison of partitions. Current available metrics are `"rand"` and `"jaccard"`
`cor_frequency`	a boolean. If `TRUE`, then computes the correlation between each partition and the total frequency of co-membership of items across all partitions. Useful to identify which partition(s) is(are) most representative of all the computed partitions.
`store_pairwise_membership`	a boolean. If `TRUE`, the pairwise membership of items is stored in the output object.
`store_confusion_matrix`	a boolean. If `TRUE`, the confusion matrices of pairwise partition comparisons are stored in the output object.

Details

This function proceeds in two main steps:

1. The first step is done within each partition. It will compare all pairs of items and document if they are clustered together (`TRUE`) or separately (`FALSE`) in each partition. For example, if site 1 and site 2 are clustered in the same cluster in partition 1, then the pairwise membership site1_site2 will be `TRUE`. The output of this first step is stored in the slot `pairwise_membership` if `store_pairwise_membership = TRUE`.
2. The second step compares all pairs of partitions by analysing if their pairwise memberships are similar or not. To do so, for each pair of partitions, the function computes a confusion matrix with four elements:

- a: number of pairs of items grouped in partition 1 and in partition 2
- b: number of pairs of items grouped in partition 1 but not in partition 2
- c: number of pairs of items not grouped in partition 1 but grouped in partition 2
- d: number of pairs of items not grouped in both partition 1 & 2

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

Based on the confusion matrices, we can compute a range of indices to indicate the agreement among partitions. As of now, we have implemented:

- *Rand index* $(a + d)/(a + b + c + d)$ The Rand index measures agreement among partitions by accounting for both the pairs of sites that are grouped, but also the pairs of sites that are not grouped.
- *Jaccard index* $(a)/(a + b + c)$ The Jaccard index measures agreement among partitions by only accounting for pairs of sites that are grouped - it is

These two metrics are complementary, because the Jaccard index will tell if partitions are similar in their clustering structure, whereas the Rand index will tell if partitions are similar not only in the pairs of items clustered together, but also in terms of the pairs of sites that are not clustered together. For example, take two partitions which never group together the same pairs of sites. Their Jaccard index will be 0, whereas the Rand index can be > 0 due to the sites that are not grouped together.

Additional indices can be manually computed by the users on the basis of the list of confusion matrices.

In some cases, users may be interested in finding which of the partitions is most representative of all partitions. To find it out, we can compare the pairwise membership of each partition with the total frequency of pairwise membership across all partitions. This correlation can be requested with `cor_frequency = TRUE`

Value

A `list` with 4 to 7 elements:

- `args`: arguments provided by the user
- `inputs`: information on the input partitions, such as the number of items being clustered
- (facultative) `pairwise_membership`: only if `store_pairwise_membership = TRUE`. This element contains the pairwise memberships of all items for each partition, in the form of a `boolean matrix` where `TRUE` means that two items are in the same cluster, and `FALSE` means that two items are not in the same cluster

- freq_item_pw_membership: A numeric vector containing the number of times each pair of items are clustered together. It corresponds to the sum of rows of the table in pairwise_membership
- (facultative) partition_freq_cor: only if cor_frequency = TRUE. A numeric vector indicating the correlation between individual partitions and the total frequency of pairwise membership across all partitions. It corresponds to the correlation between individual columns in pairwise_membership and freq_item_pw_membership
- (facultative) confusion_matrix: only if store_confusion_matrix = TRUE. A list containing all confusion matrices between each pair of partitions.
- partition_comparison: a data.frame containing the results of the comparison of partitions, where the first column indicates which partitions are compared, and the next columns correspond to the requested indices.

Author(s)

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

See Also

partition_metrics

Examples

```
# A simple case with four partitions of four items
partitions <- data.frame(matrix(nr = 4, nc = 4,
                                c(1,2,1,1,1,2,2,1,2,1,3,1,2,1,4,2),
                                byrow = TRUE))
partitions
compare_partitions(partitions)

# Find out which partitions are most representative
compare_partitions(partitions,
                   cor_frequency = TRUE)
```

---

cut_tree *Cut a hierarchical tree*

Description

This functions is designed to work on a hierarchical tree and cut it at user-selected heights. It works on either outputs from hclu_hierarclust or hclust objects. It cuts the tree for the chosen number(s) of clusters or selected height(s). It also includes a procedure to automatically return the height of cut for the chosen 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 a `hclust` object
`n_clust`	an integer or a vector of integers indicating the number of clusters to be obtained from the hierarchical tree, or the output from `partition_metrics()`. Should not be used at the same time as `cut_height`
`cut_height`	a numeric vector indicating the height(s) at which the tree should be cut. Should not be used at the same time as `n_clust` or `optim_method`
`find_h`	a boolean indicating if the height of cut should be found for the requested `n_clust`
`h_max`	a numeric indicating the maximum possible tree height for finding the height of cut when `find_h` = TRUE
`h_min`	a numeric indicating the minimum possible height in the tree for finding the height of cut when `find_h` = TRUE
`dynamic_tree_cut`	a boolean indicating if the dynamic tree cut method should be used, in which case `n_clust` & `cut_height` are ignored
`dynamic_method`	a character vector indicating the method to be used to dynamically cut the tree: either `"tree"` (clusters searched only in the tree) or `"hybrid"` (clusters searched on both tree and dissimilarity matrix)
`dynamic_minClusterSize`	an integer indicating the minimum cluster size to use in the dynamic tree cut method (see dynamicTreeCut::cutreeDynamic())
`dissimilarity`	only useful if `dynamic_method` = `"hybrid"`. Provide here the dissimilarity `data.frame` used to build the `tree`
`...`	further arguments to be passed to dynamicTreeCut::cutreeDynamic() to customize the dynamic tree cut method.

Details

The function can cut the tree with two main methods. First, it can cut the entire tree at the same height (either specified by `cut_height` or automatically defined for the chosen `n_clust`). Second, it can use the dynamic tree cut method (Langfelder et al. 2008), in which case clusters are detected with an adaptive method based on the shape of branches in the tree (thus cuts happen at multiple heights depending on cluster positions in the tree).

The dynamic tree cut method has two variants.

- The tree-based only variant (`dynamic_method = "tree"`) is a top-down approach which relies only on the tree and follows the order of clustered objects on it
- The hybrid variant (`dynamic_method = "hybrid"`) is a bottom-up approach which relies on both the tree and the dissimilarity matrix to build clusters on the basis of dissimilarity information among sites. This method is useful to detect outlying members in each cluster.

Value

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

Note

The argument `find_h` is ignored if `dynamic_tree_cut = TRUE`, because heights of cut cannot be estimated in this case.

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>), Maxime Lenormand (<`maxime.lenormand@inrae.fr`>) and 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**(5), 719–720.

See Also

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 creates a `data.frame` where each row provides one or several dissimilarity metric(s) between each pair of sites from 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 indicating which metrics to chose (see Details). Available options are *abc*, *ABC*, *Jaccard*, *Jaccardturn*, *Sorensen*, *Simpson*, *Bray*, *Brayturn* or *Euclidean*. If `"all"` is specified, then all metrics will be calculated. Can be set to `NULL` if `formula` is used.
`formula`	a `character` vector with your own formula(s) based on the `a`, `b`, `c`, `A`, `B`, and `C` quantities (see Details). `formula` is set to `NULL` by default.
`method`	a `character` indicating what method should be used to compute `abc` (see Details). `method = "prodmat"` by default is more efficient but can be greedy in memory and `method="loops"` is less efficient but less greedy in memory.

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 additional class `bioregion.pairwise.metric`, providing one or several dissimilarity metric(s) between each pair of sites. The two first columns represent each pair of sites. One column per dissimilarity metric provided in `metric` and `formula` except for the metric *abc* and *ABC* that are stored in three columns (one for each letter).

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>), Pierre Denelle (<`pierre.denelle@gmail.com`>) and 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

`similarity()` dissimilarity_to_similarity 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)

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 to 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 if the metrics based on your own formula(s) should be converted (see Details). This argument is set to `TRUE` by default.

Value

A `data.frame` with additional class `bioregion.pairwise.metric`, providing similarity metric(s) between 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`>) and Pierre Denelle (<`pierre.denelle@gmail.com`>)

See Also

similarity_to_dissimilarity() similarity() 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)

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 partitions*

Description

This function aims at optimizing one or several criteria on a set of ordered partitions. It is usually applied to find one (or several) optimal number(s) of clusters on, for example, a hierarchical tree to cut, or a range of partitions obtained from k-means or PAM. Users are advised to be careful if applied in other cases (e.g., partitions which are not ordered in an increasing or decreasing sequence, or partitions which are not related to each other).

Usage

```
find_optimal_n(
  partitions,
  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

`partitions`	a `bioregion.partition.metrics` object (output from `partition_metrics()` or a `data.frame` with the first two columns named "K" (partition name) and "n_clusters" (number of clusters) and the following columns containing evaluation metrics (numeric values)

`metrics_to_use`	character string or vector of character strings indicating upon which metric(s) in `partitions` the optimal number of clusters should be calculated. Defaults to `"all"` which means all metrics available in `partitions` will be used
`criterion`	character string indicating the criterion to be used to identify optimal number(s) of clusters. Available methods currently include `"elbow"`, `"increasing_step"`, `"decreasing_step"`, `"cutoff"`, `"breakpoints"`, `"min"` or `"max"`. Default is `"elbow"`. See details.
`step_quantile`	if `"increasing_step"` or `"decreasing_step"`, specify here the quantile of differences between two consecutive k to be used as the cutoff to identify the most important steps in `eval_metric`
`step_levels`	if `"increasing_step"` or `"decreasing_step"`, specify here the number of largest steps to keep as cutoffs.
`step_round_above`	a `boolean` indicating if the optimal number of clusters should be picked above or below the identified steps. Indeed, each step will correspond to a sudden increase or decrease between partition X & partition X+1: should the optimal partition be X+1 (`step_round_above = TRUE`) or X (`step_round_above = FALSE`? Defaults to `TRUE`
`metric_cutoffs`	if `criterion = "cutoff"`, specify here the cutoffs of `eval_metric` at which the number of clusters should be extracted
`n_breakpoints`	specify here the number of breakpoints to look for in the curve. Defaults to 1
`plot`	a boolean indicating if a plot of the first `eval_metric` should be drawn with the identified optimal numbers of cutoffs

Details

This function explores the relationship evaluation metric ~ number of clusters, and a criterion is applied to search an optimal number of clusters.

Please read the note section about the following criteria.

Foreword:

Here we implemented a set of criteria commonly found in the literature or recommended in the bioregionalisation literature. Nevertheless, we also advocate to move beyond the "Search one optimal number of clusters" paradigm, and consider investigating "multiple optimal numbers of clusters". Indeed, using only one optimal number of clusters may simplify the natural complexity of biological datasets, and, for example, ignore the often hierarchical / nested nature of bioregionalisations. Using multiple partitions likely avoids this oversimplification bias and may convey more information. See, for example, the reanalysis of Holt et al. (2013) by (Ficetola et al. 2017), where they used deep, intermediate and shallow cuts.

Following this rationale, several of the criteria implemented here can/will return multiple "optimal" numbers of clusters, depending on user choices.

Criteria to find optimal number(s) of clusters

- `elbow`: This method consists in finding one elbow in the evaluation metric curve, as is commonly done in clustering analyses. The idea is to approximate the number of clusters at which the evaluation metric no longer increments.It is based on a fast method finding the maximum distance between the curve and a straight line linking the minimum and maximum number of points. The code we use here is based on code written by Esben Eickhardt available here https://stackoverflow.com/questions/2018178/finding-the-best-trade-off-point-on-a-curve/42810075#42810075. The code has been modified to work on both increasing and decreasing evaluation metrics.

- `increasing_step` or `decreasing_step`: This method consists in identifying clusters at the most important changes, or steps, in the evaluation metric. The objective can be to either look for largest increases (`increasing_step`) or largest decreases `decreasing_step`. Steps are calculated based on the pairwise differences between partitions. Therefore, this is relative to the distribution of differences in the evaluation metric over the tested partitions. Specify `step_quantile` as the quantile cutoff above which steps will be selected as most important (by default, 0.99, i.e. the largest 1\ selected).Alternatively, you can also choose to specify the number of top steps to keep, e.g. to keep the largest three steps, specify `step_level = 3`. Basically this method will emphasize the most important changes in the evaluation metric as a first approximation of where important cuts can be chosen.

  **Please note that you should choose between `increasing_step` and `decreasing_step` depending on the nature of your evaluation metrics. For example, for metrics that are monotonously decreasing (e.g., endemism metrics `"avg_endemism"` & `"tot_endemism"`) with the number of clusters should n_clusters, you should choose `decreasing_step`. On the contrary, for metrics that are monotonously increasing with the number of clusters (e.g., `"pc_distance"`), you should choose `increasing_step`. **
- `cutoffs`: This method consists in specifying the cutoff value(s) in the evaluation metric from which the number(s) of clusters should be derived. This is the method used by (Holt et al. 2013). Note, however, that the cut-offs suggested by Holt et al. (0.9, 0.95, 0.99, 0.999) may be only relevant at very large spatial scales, and lower cut-offs should be considered at finer spatial scales.
- `breakpoints`: This method consists in finding break points in the curve using a segmented regression. Users have to specify the number of expected break points in `n_breakpoints` (defaults to 1). Note that since this method relies on a regression model, it should probably not be applied with a low number of partitions.
- `min` & `max`: Picks the optimal partition(s) respectively at the minimum or maximum value of the evaluation metric.

Value

a `list` of class `bioregion.optimal.n` with three elements:

- `args`: input arguments
- `evaluation_df`: the input evaluation data.frame appended with `boolean` columns identifying the optimal numbers of clusters
- `optimal_nb_clusters`: a list containing the optimal number(s) of cluster(s) for each metric specified in `"metrics_to_use"`, based on the chosen `criterion`
- `plot`: if requested, the plot will be stored in this slot

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 bioregionalisation.

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>), Maxime Lenormand (<`maxime.lenormand@inrae.fr`>) and 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**(1), 9–11.

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

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**(6115), 74–78.

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

Langfelder P, Zhang B, Horvath S (2008). "Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R." *BIOINFORMATICS*, **24**(5), 719–720.

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 = 2:15,
                          index = "Simpson")
tree1

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

```
find_optimal_n(a)
find_optimal_n(a, criterion = "increasing_step")
find_optimal_n(a, criterion = "decreasing_step")
find_optimal_n(a, criterion = "decreasing_step",
               step_levels = 3)
find_optimal_n(a, criterion = "decreasing_step",
               step_quantile = .9)
find_optimal_n(a, criterion = "decreasing_step",
               step_levels = 3)
find_optimal_n(a, criterion = "decreasing_step",
               step_levels = 3)
find_optimal_n(a, criterion = "breakpoints")
```

`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 get 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` and then `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 next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`n_clust`	an `integer` or an `integer` vector indicating the number of clusters to be obtained from the hierarchical tree, or the output from partition_metrics. Should not be used at the same time as `cut_height`.
`cut_height`	a `numeric` vector indicating the height(s) at which the tree should be cut. Should not be used at the same time as `n_clust`.
`find_h`	a `boolean` indicating if the height of cut should be found for the requested `n_clust`.
`h_max`	a `numeric` indicating the maximum possible tree height for the chosen `index`.
`h_min`	a `numeric` indicating the minimum possible height in the tree for the chosen `index`.

Details

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

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

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
5. **clusters**: `data.frame` containing the clustering results

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>), Boris Leroy (<`leroy.boris@gmail.com`>) and 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

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 can get 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` and then `similarity_to_dissimilarity`.

Usage

```
hclu_hierarclust(
  dissimilarity,
  index = names(dissimilarity)[3],
  method = "average",
  randomize = TRUE,
  n_runs = 30,
  keep_trials = FALSE,
  optimal_tree_method = "best",
  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 next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`method`	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 if the dissimilarity matrix should be randomized, to account for the order of sites in the dissimilarity matrix.
`n_runs`	number of trials to randomize the dissimilarity matrix.
`keep_trials`	a `boolean` indicating if all random trial results. should be stored in the output object (set to FALSE to save space if your `dissimilarity` object is large).
`optimal_tree_method`	a `character` indicating how the final tree should be obtained from all trials. The only option currently is "best", which means the tree with the best cophenetic correlation coefficient will be chosen.
`n_clust`	an `integer` or an `integer` vector indicating the number of clusters to be obtained from the hierarchical tree, or the output from partition_metrics. Should not be used at the same time as `cut_height`.
`cut_height`	a `numeric` vector indicating the height(s) at which the tree should be cut. Should not be used at the same time as `n_clust`.
`find_h`	a `boolean` indicating if the height of cut should be found for the requested `n_clust`.
`h_max`	a `numeric` indicating the maximum possible tree height for the chosen `index`.
`h_min`	a `numeric` indicating the minimum possible height in the tree for the chosen `index`.

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 and 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 `partition_metrics()`

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
5. **clusters**: `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 contains the dissimilarity matrix, with site order randomized, the associated tree and the cophenetic correlation coefficient (Spearman) for that tree
- `final.tree`: a `hclust` object containing the final hierarchical tree to be used
- `final.tree.coph.cor`: the cophenetic correlation coefficient between the initial dissimilarity matrix and `final.tree`

Author(s)

Boris Leroy (<leroy.boris@gmail.com>), Pierre Denelle (<pierre.denelle@gmail.com>) and 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.

See Also

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")

# 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)

# Recut the tree afterwards
tree3.1 <- cut_tree(tree3, n = 5)

tree4 <- hclu_hierarclust(dissim, n_clust = 1:19)
```

`hclu_optics` *OPTICS hierarchical clustering algorithm*

Description

This function performs semi-hierarchical clustering on the basis of dissimilarity with 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 next column(s) are the dissimilarity indices.

`index`	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 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 is 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 data set 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 if the hierarchy should be pruned out from the output to only keep clusters at the "minimal" level, i.e. only leaf / non-overlapping clusters. If `TRUE`, then argument `show_hierarchy` should be `FALSE`.
`show_hierarchy`	a `boolean` specifying if 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 visualising the plot of the OPTICS object. If `show_hierarchy = TRUE`, then the output cluster `data.frame` will contain additional columns showing the hierarchy of clusters.
`algorithm_in_output`	a `boolean` indicating if the original output of dbscan should be returned in the output (`TRUE` by default, see Value).
`...`	you can add here further 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**: `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
5. **clusters**: `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>) and 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). ISSN 15487660.

See Also

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 permission and test the bioregion's binary files*

Description

This function downloads and unzips the 'bin' folder needed to run some functions of bioregion. It also checks if the files have the permissions to be executed as programs. It finally tests if the binary files are running properly.

Usage

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

Arguments

`binpath`	a `character` indicating the path to the folder that will host the 'bin' folder containing the binary files (see Details).
`infomap_version`	a `character` vector indicating 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's package folder (`binpath = "pkgfolder"`). Finally, a path to a folder of your choice can be chosen.

In any case, PLEASE MAKE SURE to update the binpath accordingly in netclu_infomap, netclu_louvain and netclu_oslom).

Value

No return value

Note

Only the Infomap version 2.1.0, 2.6.0 and 2.7.1 are available for now.

Author(s)

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

`map_clusters` *Create a map of bioregions*

Description

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

Usage

```
map_clusters(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 next column(s) 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 in `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` is cluster is a `bioregion.clusters` object) and the site 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`>) and Pierre Denelle (<`pierre.denelle@gmail.com`>)

Examples

```
data(fishmat)
data(fishsf)

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

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

Description

This function creates a two- or three-columns `data.frame` where each row represents the interaction between two nodes (site and species for example) and an optional third column indicating the weight of the interaction (if `weight = TRUE`) from a contingency table (sites as rows and species as columns for example).

Usage

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

Arguments

`mat`	a contingency table (i.e. `matrix`).
`weight`	a `boolean` indicating if the value are weights.
`remove_zeroes`	a `boolean` determining whether interactions with weight equal to 0 should be removed from the output.
`include_diag`	a `boolean` indicating whether the diagonal should be included in the output. Only for squared matrix.
`include_lower`	a `boolean` indicating whether the lower triangular matrix should be included in the output. Only for squared matrix.

Value

A `data.frame` where each row represents the interaction between two nodes and an optional third column indicating the weight of the interaction.

Author(s)

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

See Also

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 network 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 to it.

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 two first columns as undirected links between pair of nodes and and the next column(s) are the weight of the links.
`weight`	a `boolean` indicating if the weights should be considered if there are more than two columns (see Note).
`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 (O 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).
`forceLPA`	a `boolean` indicating if 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`	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 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**: `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 computeModules.

Note

Beckett has been designed to deal with weighted bipartite networks. Note that if `weight = FALSE`, a weight of 1 will be assigned to each pair of nodes. 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`>) and Boris Leroy (<`leroy.boris@gmail.com`>)

References

Beckett SJ (2016). "Improved community detection in weighted bipartite networks." *Royal Society Open Science*, **3**(1), 140536.

See Also

netclu_infomap, 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 (O by default).
`index`	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`	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_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`>) and 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.

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 = "2.7.1",
  binpath = "tempdir",
  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 (O 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).
`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 results in fewer modules (default is 1).
`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 (default is 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`	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).
`version`	a `character` indicating the Infomap version to use.
`binpath`	a `character` indicating the path to the bin folder (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 and 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.

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

Several version 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**: `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
5. **clusters**: `data.frame` containing the clustering results

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

- `cmd`: the command line use 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 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, `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`>) and 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**(4), 1118–1123.

See Also

install_binaries, 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 (O 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).
`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_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**: `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 a "communities" object, output of cluster_label_prop.

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`>) and 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**(3), 036106.

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 leading eigen vector of the community matrix*

Description

This function finds communities in a (un)weighted undirected network based on leading eigen vector 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 (O by default).
`index`	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`	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_leading_eigen should be returned in the output (`TRUE` by default, see Value).

Details

This function is based on 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**: `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_leading_eigen.

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`>) and Boris Leroy (<`leroy.boris@gmail.com`>)

References

Newman MEJ (2006). “Finding community structure in networks using the eigenvectors of matrices.” *Physical Review E*, **74**(3), 036104.

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`	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).
`objective_function`	a string indicating the objective function to use, the Constant Potts Model ("CPM") or "modularity" ("CPM" by default).
`resolution_parameter`	the resolution parameter to use. Higher resolutions lead to more smaller communities, while lower resolutions lead to fewer larger communities.
`beta`	parameter affecting the randomness in the Leiden algorithm. This affects only the refinement step of the algorithm.
`n_iterations`	the number of iterations to iterate the Leiden algorithm. Each iteration may improve the partition further.
`vertex_weights`	the vertex weights used in the Leiden algorithm. If this is not provided, it will be automatically determined on the basis of the objective_function. Please see the details of this function how to interpret the vertex weights.
`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 ("sites", "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_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**: `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_leiden.

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>) and 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**(1), 5233. Publisher: Nature Publishing Group UK London.

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",
  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 (O by default).
`index`	name or number of the column to use as weight. By default, the third column name of `net` is used.
`lang`	a string indicating what 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`	for the random number generator (only when `lang = "igraph"`, NULL for random by default).
`q`	the quality function used to compute 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`	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).
`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 proposed two implementations of the function (parameter `lang`): the igraph implementation (cluster_louvain) and the C++ implementation (`https://sourceforge.net/projects/louvain/`, version 0.3).

The igraph implementation offers the possibility to adjust the resolution parameter of the modularity function (`resolution` argument) that the algorithm uses internally. 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 offers the possibility to choose among several quality functions, `q = 0` for the classical Newman-Girvan criterion (also called "Modularity"), 1 for the Zahn-Condorcet criterion, 2 for the Owsinski-Zadrozny criterion (you should specify the value of the parameter with the `c` argument), 3 for the Goldberg Density criterion, 4 for the A-weighted Condorcet criterion, 5 for the Deviation to Indetermination criterion, 6 for the Deviation to Uniformity criterion, 7 for the Profile Difference criterion, 8 for the Shi-Malik criterion (you should specify the value of kappa_min with `k` argument) and 9 for the Balanced Modularity criterion.

The C++ version of Louvain is based on the version 0.3 (`https://sourceforge.net/projects/louvain/`). 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.

The C++ version of Louvain generates temporary folders and/or files that are stored in the `path_temp` folder ("louvain_temp" with an 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**: `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 an the output of cluster_louvain if `lang = "igraph"` and the following element if `lang = "cpp"`:

- `cmd`: the command line use to run Louvain
- `version`: the Louvain version
- `web`: Louvain's website

.

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`>) and 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*, P10008.

See Also

`install_binaries()`, `netclu_infomap()`, `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",
  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 (O 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 reassign 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 you to get more modules.
`cp`	kind of resolution parameter used to decide between taking some modules or their union (default value is 0.5, 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).
`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.

The C++ version of OSLOM generates temporary folders and/or files that are stored in the `path_temp` folder (folder "oslom_temp" with an 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**: `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
5. **clusters**: `data.frame` containing the clustering results

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

- `cmd`: the command line use 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 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.

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 were found for this node (i.e. non-overlapping nodes).

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>), Pierre Denelle (<pierre.denelle@gmail.com>) and 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**(4).

See Also

`install_binaries()`, `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 (O 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 and Latapy 2005) as implemented in the igraph package (cluster_walktrap).

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_walktrap.

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`>) and 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.

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 creates a contingency table from a two- or three-columns `data.frame` where each row represents the interaction between two nodes (site and species for example) and an optional third column indicating 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-columns `data.frame` where each row represents the interaction between two nodes (site and species for example) and an optional third column indicating the weight of the interaction.
`weight`	a `boolean` indicating if the weight should be considered
`squared`	a `boolean` indicating if the output matrix should but squared (same nodes in rows and columns).
`symmetrical`	a `boolean` indicating if the resulting matrix should be symmetrical (only if `squared = TRUE`). Note that different weights associated with two opposite pairs already present in net will be preserved.
`missing_value`	the value to assign to the pairs of nodes not present in net (0 by default).

Value

A `matrix` with the first nodes (first column of `net`) as rows and the second nodes (second column of `net`) as columns. Note that if `squared = TRUE` the rows and columns have the same number of elements corresponding to the concatenation of unique objects in `net`'s first and second columns. If `squared = TRUE` the matrix can 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`>) and Boris Leroy (<`leroy.boris@gmail.com`>)

See Also

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_clara *Non hierarchical clustering: CLARA*

Description

This function performs non hierarchical clustering on the basis of dissimilarity with partitioning around medoids, using 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 represent pairs of sites (or any pair of nodes), and the next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`seed`	for the random number generator (NULL for random by default).

`n_clust`	an `integer` or an `integer` vector specifying the requested number(s) of clusters.
`maxiter`	an `integer` defining the maximum number of iterations.
`initializer`	a `character`, either 'BUILD' (used in classic PAM algorithm) or 'LAB' (linear approximative BUILD).
`fasttol`	positive `numeric` defining the tolerance for fast swapping behavior, set to 1 by default.
`numsamples`	positive `integer` defining the number of samples to draw.
`sampling`	positive `numeric` defining the sampling rate.
`independent`	a `boolean` indicating that the previous medoids are not kept in the next sample (FALSE by default).
`algorithm_in_output`	a `boolean` indicating if the original output of fastclara should be returned in the output (`TRUE` by default, see Value).

Details

Based on fastkmedoids package (fastclara).

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 fastclara.

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>), Boris Leroy (<`leroy.boris@gmail.com`>), and 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

nhclu_pam

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_clara(dissim, index = "Simpson", n_clust = 5)

partition_metrics(clust1, dissimilarity = dissim,
eval_metric = "pc_distance")
```

`nhclu_clarans` *Non hierarchical clustering: CLARANS*

Description

This function performs non hierarchical clustering on the basis of dissimilarity with partitioning around medoids, using 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 represent pairs of sites (or any pair of nodes), and the next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`seed`	for the random number generator (NULL for random by default).
`n_clust`	an `integer` or an `integer` vector specifying the requested number(s) of clusters.
`numlocal`	an `integer` defining the number of samples to draw.

`maxneighbor`	a positive `numeric` defining the sampling rate.
`algorithm_in_output`	a `boolean` indicating if the original output of fastclarans should be returned in the output (`TRUE` by default, see Value).

Details

Based on fastkmedoids package (fastclarans).

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
5. **clusters**: `data.frame` containing the clustering results

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

Author(s)

Pierre Denelle (<`pierre.denelle@gmail.com`>), Boris Leroy (<`leroy.boris@gmail.com`>), and 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

nhclu_pam

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_clarans(dissim, index = "Simpson", n_clust = 5)

partition_metrics(clust1, dissimilarity = dissim,
eval_metric = "pc_distance")
```

nhclu_dbscan *dbscan clustering*

Description

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

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 represent pairs of sites (or any pair of nodes), and the next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`minPts`	a `numeric` value or a `numeric` vector 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` value or a `numeric` vector specifying the eps argument of dbscan). eps specifies how similar points should be to each other to be considered a part of a cluster. See details for guidance on choosing this parameter.
`plot`	a `boolean` indicating if the k-nearest neighbor distance plot should be plotted.
`algorithm_in_output`	a `boolean` indicating if the original output of dbscan should be returned in the output (`TRUE` by default, see Value).
`...`	you can add here further arguments to be passed to `dbscan()` (see dbscan).

Details

The dbscan (Density-based spatial clustering of applications with noise) clustering algorithm clusters points on the basis of the density of neighbours around each data points. It necessitates two main arguments, `minPts`, which stands for the minimum number of points to identify a core, and `eps`, which is the radius to find neighbors. `minPts` and `eps` should be defined by the user, which is not straightforward. We recommend reading the help in dbscan) to learn how to set these arguments, as well as the paper (Hahsler et al. 2019). Note that clusters with a value of 0 are points which were deemed as noise by the algorithm.

By default the function will select values for `minPts` and `eps`. However, these values can be inadequate and the users is advised to tune these values by running the function multiple times.

Choosing minPts: how many points should be necessary to make a cluster? i.e., what is the minimum number of sites you expect in a bioregion? Set a value sufficiently large for your dataset and your expectations.

Choosing eps: how similar should sites be in a cluster? If `eps` is too small, then a majority of points will be considered too distinct and will not be clustered at all (i.e., considered as noise)? If the value is too high, then clusters will merge together. The value of `eps` depends on the `minPts` argument, and the literature recommends to choose `eps` by identifying a knee in the k-nearest neighbor distance plot. By default the function will try to automatically find a knee in that curve, but the result is uncertain, and so the user should inspect the graph and modify `dbscan_eps` accordingly. To explore `eps` values, follow the recommendation by the function when you launch it a first time without defining `eps`. Then, adjust depending on your clustering results.

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
5. **clusters**: `data.frame` containing the clustering results

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

Author(s)

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

See Also

hclu_optics

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 on the basis of dissimilarity with 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 represent pairs of sites (or any pair of nodes), and the next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`seed`	for the random number generator (NULL for random by default).
`n_clust`	an `integer` or an `integer` vector specifying the requested number(s) of clusters
`iter_max`	an `integer` specifying the maximum number of iterations for the kmeans method (see kmeans)
`nstart`	an `integer` specifying how many random sets of `n_clust` should be selected as starting points for the kmeans analysis (see kmeans)
`algorithm`	a `character` specifying the algorithm to use for kmean (see kmeans). Available options are Hartigan-Wong, Lloyd, Forgy and MacQueen.
`algorithm_in_output`	a `boolean` indicating if the original output of kmeans should be returned in the output (`TRUE` by default, see Value).

Details

This method partitions the data into k groups such that that the sum of squares of euclidean distances from points to the assigned cluster centers is minimized. k-means cannot be applied directly on dissimilarity/beta-diversity metrics, because these distances are not euclidean. Therefore, it requires first to transform the dissimilarity matrix with a Principal Coordinate Analysis (using the function pcoa), and then applying k-means on the coordinates of points in the PCoA. Because this makes an additional transformation of the initial matrix of dissimilarity, the partitioning around medoids method should be preferred (nhclu_pam)

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
5. **clusters**: `data.frame` containing the clustering results

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

Author(s)

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

See Also

nhclu_pam

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")

clust1 <- nhclu_kmeans(dissim, n_clust = 2:10, index = "Simpson")
clust2 <- nhclu_kmeans(dissim, n_clust = 2:15, index = "Simpson")
partition_metrics(clust2, dissimilarity = dissim,
                  eval_metric = "pc_distance")

partition_metrics(clust2, net = comnet, species_col = "Node2",
                  site_col = "Node1", eval_metric = "avg_endemism")
```

nhclu_pam *Non hierarchical clustering: partitioning around medoids*

Description

This function performs non hierarchical clustering on the basis of dissimilarity with partitioning around medoids.

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 represent pairs of sites (or any pair of nodes), and the next column(s) are the dissimilarity indices.
`index`	name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`seed`	for the random number generator (NULL for random by default).
`n_clust`	an `integer` or an `integer` vector specifying the requested number(s) of clusters.
`variant`	a `character` string specifying the variant of pam to use, by default `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 start for the pam algorithm. By default, 1 (for the `faster` variant).
`cluster_only`	a `boolean` specifying if only the clustering should be returned from the pam function (more efficient).
`algorithm_in_output`	a `boolean` indicating if the original output of pam should be returned in the output (`TRUE` by default, see Value).
`...`	you can add here further arguments to be passed to `pam()` (see pam)

Details

This method partitions data into the chosen number of cluster on the basis of the input dissimilarity matrix. It is more robust than k-means because it minimizes the sum of dissimilarity between cluster centres and points assigned to the cluster - whereas the k-means approach minimizes the sum of squared euclidean distances (thus k-means cannot be applied directly on the input dissimilarity matrix if the distances are not euclidean).

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
5. **clusters**: `data.frame` containing the clustering results

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

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>), Pierre Denelle (<`pierre.denelle@gmail.com`>) and 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

nhclu_kmeans

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")

clust1 <- nhclu_pam(dissim, n_clust = 2:10, index = "Simpson")
clust2 <- nhclu_pam(dissim, n_clust = 2:15, index = "Simpson")
partition_metrics(clust2, dissimilarity = dissim,
eval_metric = "pc_distance")
partition_metrics(clust2, net = comnet, species_col = "Node2",
                   site_col = "Node1", eval_metric = "avg_endemism")
```

partition_metrics *Calculate metrics for one or several partitions*

Description

This function aims at calculating metrics for one or several partitions, usually on outputs from `netclu_`, `hclu_` or `nhclu_` functions. Metrics may require the users to provide either a similarity or dissimilarity matrix, or to provide the initial species-site table.

Usage

```
partition_metrics(
  cluster_object,
  dissimilarity = NULL,
  dissimilarity_index = NULL,
  net = NULL,
  site_col = 1,
  species_col = 2,
  eval_metric = c("pc_distance", "anosim", "avg_endemism", "tot_endemism")
)
```

Arguments

`cluster_object`	a `bioregion.clusters` object
`dissimilarity`	a `dist` object or a `bioregion.pairwise.metric` object (output from `similarity_to_dissimilarity(` Necessary 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 be used in case `dist` is a `data.frame` with multiple dissimilarity indices
`net`	the species-site network (i.e., bipartite network). Should be provided if `eval_metric` includes `"avg_endemism"` or `"tot_endemism"`
`site_col`	name or number for the column of site nodes (i.e. primary nodes). Should be provided if `eval_metric` includes `"avg_endemism"` or `"tot_endemism"`
`species_col`	name or number for the column of species nodes (i.e. feature nodes). Should be provided if `eval_metric` includes `"avg_endemism"` or `"tot_endemism"`
`eval_metric`	character string or vector of character strings indicating metric(s) to be calculated to investigate the effect of different number of clusters. Available options: `"pc_distance"`, `"anosim"`, `"avg_endemism"` and `"tot_endemism"`

Details

Evaluation metrics:

- `pc_distance`: this metric is the method used by (Holt et al. 2013). It is a ratio of the between-cluster sum of dissimilarity (beta-diversity) versus the total sum of dissimilarity (beta-diversity) for the full dissimilarity matrix. In other words, it is calculated on the basis of two elements. First, the total sum of dissimilarity is calculated by summing the entire dissimilarity matrix (`dist`). Second, the between-cluster sum of dissimilarity is calculated as follows: for a given number of cluster, the dissimilarity is only summed between clusters, not within clusters. To do that efficiently, all pairs of sites within the same clusters have their dissimilarity set to zero in the dissimilarity matrix, and then the dissimilarity matrix is summed. The `pc_distance` ratio is obtained by dividing the between-cluster sum of dissimilarity by the total sum of dissimilarity.
- `anosim`: This metric is the statistic used in Analysis of Similarities, as suggested in (Castro-Insua et al. 2018) (see vegan::anosim()). It compares the between-cluster dissimilarities to the within-cluster dissimilarities. It is based based on the difference of mean ranks between groups and within groups with the following formula: $R = (r_B - r_W)/(N(N-1)/4)$, where r_B and r_W are the average ranks between and within clusters respectively, and N is the total number of sites. Note that the function does not estimate the significance here, it only computes the statistic - for significance testing see vegan::anosim().
- `avg_endemism`: this metric is the average percentage of endemism in clusters as recommended by (Kreft and Jetz 2010). Calculated as follows: $End_{mean} = \frac{\sum_{i=1}^{K} E_i / S_i}{K}$ where E_i is the number of endemic species in cluster i, S_i is the number of species in cluster i, and K the maximum number of clusters.
- `tot_endemism`: this metric is the total endemism across all clusters, as recommended by (Kreft and Jetz 2010). Calculated as follows: $End_{tot} = \frac{E}{C}$
  where E is total the number of endemics (i.e., species found in only one cluster) and C is the number of non-endemic species.

Value

a `list` of class `bioregion.partition.metrics` with two to three elements:

- `args`: input arguments
- `evaluation_df`: the data.frame containing `eval_metric` for all explored numbers of clusters
- `endemism_results`: if endemism calculations were requested, a list with the endemism results for each partition

Author(s)

Boris Leroy (<`leroy.boris@gmail.com`>), Maxime Lenormand (<`maxime.lenormand@inrae.fr`>) and 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**(1), 9–11.

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

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**(6115), 74–78.

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

Langfelder P, Zhang B, Horvath S (2008). "Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R." *BIOINFORMATICS*, **24**(5), 719–720.

See Also

compare_partitions

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 = 2:20, index = "Simpson")
tree1

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

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

Description

This function creates a `data.frame` where each row provides one or several similarity metric(s) between each pair of sites from 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 indicating which metrics to chose (see Details). Available options are *abc*, *ABC*, *Jaccard*, *Jaccardturn*, *Sorensen*, *Simpson*, *Bray*, *Brayturn* or *Euclidean*. If `"all"` is specified, then all metrics will be calculated. Can be set to `NULL` if `formula` is used.
`formula`	a `character` vector with your own formula(s) based on the `a`, `b`, `c`, `A`, `B`, and `C` quantities (see Details). `formula` is set to `NULL` by default.
`method`	a string indicating what method should be used to compute `abc` (see Details). `method = "prodmat"` by default is more efficient but can be greedy in memory and `method = "loops"` is less efficient but less greedy in memory.

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 additional class `bioregion.pairwise.metric`, providing one or several similarity metric(s) between each pair of sites. The two first columns represent each pair of sites. One column per similarity metric provided in `metric` and `formula` except for the metric *abc* and *ABC* that are stored in three columns (one for each letter).

Author(s)

Maxime Lenormand (<maxime.lenormand@inrae.fr>), Pierre Denelle (<pierre.denelle@gmail.com>) and 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

dissimilarity dissimilarity_to_similarity 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 to 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 if the metrics based on your own formula(s) should be converted (see Details). This argument is set to `TRUE` by default.

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:

$Euclideandistance = (1 - Euclideansimilarity)/Euclideansimilarity$

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`>) and Pierre Denelle (<`pierre.denelle@gmail.com`>)

See Also

`dissimilarity_to_similarity()` `similarity()` `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
```

`subset_node` *Extract a subset of nodes from a bioregion.clusters object*

Description

This function extracts a subset of nodes according to its type (sites or species) from a bioregion.clusters object containing both types of nodes (sites and species).

Usage

```
subset_node(clusters, node_type = "sites")
```

Arguments

`clusters`	an object of class `bioregion.clusters`.
`node_type`	a `character` indicating what types of nodes ("sites" or "species") should be extracted (`node_type = "sites"` by default).

Value

An object of class `bioregion.clusters` with a given node type (sites or species).

Note

The network clustering functions (prefix `netclu_`) may return both types of nodes (sites and species) when applied on bipartite networks (argument `bipartite`). In this case, the type of nodes returned in the output can be chosen with the argument `return_node_type`. This function allows to retrieve a particular type of nodes (sites or species) from the output and modify the return_node_type accordingly.

Author(s)

Maxime Lenormand (<`maxime.lenormand@inrae.fr`>), Pierre Denelle (<`pierre.denelle@gmail.com`>) and 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 <- subset_node(clusters, node_type = "sites")
```

`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