Package 'bioregion'

October 10, 2026

Type Package

Title Comparison of Bioregionalization Methods

Version 1.5.0

Description The main purpose of this package is to propose a transparent methodological framework to compare bioregionalization 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, apcluster, bipartite, cluster, data.table, dbscan, dynamicTreeCut, fastcluster, fastkmedoids, ggplot2, grDevices, httr, igraph, mathjaxr, Matrix, phangorn, rcartocolor, Rdpack, rlang, rmarkdown, segmented, sf, stats, tidyr, utils, vegan

RdMacros mathjaxr, Rdpack

LinkingTo Rcpp

Suggests ade4, adespatial, betapart, dplyr, ecodist, knitr, microbenchmark, parallel, rnaturalearth, rnaturalearthdata, terra, testthat (>= 3.0.0)

VignetteBuilder knitr

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

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

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation yes

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

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

Repository CRAN

Date/Publication 2026-10-10 18:10:02 UTC

Contents

as_bioregion_pairwise . . . 3
betapart_to_bioregion . . . 5
bind_pairwise . . . 6
bioregionalization_metrics . . . 7
bioregion_colors . . . 9
bioregion_metrics . . . 12
compare_bioregionalizations . . . 14
cut_tree . . . 17
dissimilarity . . . 19
dissimilarity_to_similarity . . . 21
exportGDF . . . 23
find_optimal_n . . . 26
fishdf . . . 29
fishmat . . . 29
fishsf . . . 30
hclu_diana . . . 30
hclu_hierarclust . . . 32
hclu_optics . . . 39
ihct . . . 41
install_binaries . . . 47
map_bioregions . . . 48
mat_to_net . . . 50
netclu_beckett . . . 52
netclu_greedy . . . 54
netclu_infomap . . . 56
netclu_labelprop . . . 59
netclu_leadingeigen . . . 61
netclu_leiden . . . 64
netclu_louvain . . . 66
netclu_oslom . . . 70
netclu_walktrap . . . 73
net_to_mat . . . 75
nhclu_affprop . . . 76
nhclu_clara . . . 79
nhclu_clarans . . . 81
nhclu_dbscan . . . 83
nhclu_kmeans . . . 85

nhclu_pam 87
similarity 89
similarity_to_dissimilarity 91
site_species_metrics 93
site_species_subset 98
vegedf 99
vegemat 99
vegesf 100

Index **101**

`as_bioregion_pairwise` *Convert a matrix or list of matrices to a bioregion (dis)similarity object*

Description

Converts a (dis)similarity `matrix` or a `list` of such matrices into a `bioregion.pairwise` object compatible with the `bioregion` package. The input can come from base R, `dist` objects, or outputs from other packages.

Usage

```
as_bioregion_pairwise(
  mat,
  metric_name = NULL,
  pkg = NULL,
  is_similarity = FALSE
)
```

Arguments

`mat`	A `matrix`, a `dist` object, or a `list` of these representing pairwise similarity or dissimilarity values to convert into a `bioregion.pairwise` object. This function can also directly handle outputs from other R packages (see the `pkg` argument).
`metric_name`	Optional `character` vector or single `character` string specifying the name of the (dis)similarity metric(s), which will appear as column names in the output (see Note).
`pkg`	An optional `character` string indicating the name of the package from which `mat` was generated (`NULL` by default, see Details). Available options are `"adespatial"`, `"betapart"`, `"ecodist"`, or `"vegan"`.
`is_similarity`	A `logical` value indicating whether the input data represents similarity (`TRUE`) or dissimilarity (`FALSE`).

Details

This function can directly handle outputs from ten functions across four packages:

- **adespatial**: beta.div, beta.div.comp
- **betapart**: beta.pair, beta.pair.abund, betapart.core, betapart.core.abund
- **ecodist**: distance, bcdist
- **vegan**: vegdist, designdist

See the documentation of these packages for more information:

- https://cran.r-project.org/package=adespatial
- https://cran.r-project.org/package=betapart
- https://cran.r-project.org/package=ecodist
- https://cran.r-project.org/package=vegan

Value

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

Note

If no specific package is specified (i.e., `pkg = NULL`), site names will be based on the row names of the first matrix. If row names are `NULL`, they will be generated automatically. If `mat` is a named list, those names will be used as column names only if `metric_name = NULL`.

Author(s)

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

See Also

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

Associated functions: dissimilarity similarity bind_pairwise

Examples

```
mat <- matrix(runif(100), 10, 10)
rownames(mat) <- paste0("s",1:10)

pair <- as_bioregion_pairwise(list(mat,mat,mat),
                              metric_name = NULL,
                              pkg = NULL,
                              is_similarity = FALSE)
```

`betapart_to_bioregion` *Convert betapart dissimilarity to bioregion dissimilarity (DEPRECATED)*

Description

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

Usage

```
betapart_to_bioregion(betapart_result)
```

Arguments

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

Value

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

Author(s)

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

See Also

This function is deprecated, use as_bioregion_pairwise instead.

Examples

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

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

## End(Not run)
```

bind_pairwise *Combine and enrich bioregion (dis)similarity object(s)*

Description

Combine two bioregion.pairwise objects and/or compute new pairwise metrics based on the columns of the object(s).

Usage

```
bind_pairwise(primary_metrics, secondary_metrics, new_metrics = NULL)
```

Arguments

primary_metrics	A bioregion.pairwise object. This is the main set of pairwise metrics that will be used as a base for the combination.
secondary_metrics	A second bioregion.pairwise object to be combined with primary_metrics. It must have the same sites identifiers and pairwise structure. Can be set to NULL if new_metrics is specified.
new_metrics	A character vector or a single character string specifying custom formula(s) based on the column names of primary_metrics and secondary_metrics (see Details).

Details

When both primary_metrics and secondary_metrics are provided and if the pairwise structure is identical the function combine the two objects. If new_metrics is provided, each formula is evaluated based on the column names of primary_metrics (and secondary_metrics if provided).

Value

A new bioregion.pairwise object containing the combined and/or enriched data. It includes all original metrics from the inputs, as well as any newly computed metrics.

Author(s)

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

See Also

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

Associated functions: dissimilarity similarity as_bioregion_pairwise

Examples

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

sim <- bind_pairwise(primary_metrics = similarity(comat,
                                                  metric = "abc"),
                                  secondary_metrics = similarity(comat,
                                                    metric = "Simpson"),
                                  new_metrics = "1 - (b + c) / (a + b + c)")
```

`bioregionalization_metrics`

Calculate metrics for a bioregionalization

Description

This function calculates metrics at the bioregionalization level. These evaluation metrics can be used to assess and select a partition based on the optimal number of clusters.

Usage

```
bioregionalization_metrics(
  bioregionalization,
  eval_metrics = c("prop_between_dissim", "anosim"),
  dissimilarity,
  dissimilarity_index = names(dissimilarity)[3],
  comat = NULL,
  anosim_permutations = 1,
  eval_metric = NULL,
  net = NULL,
  site_col = NULL,
  species_col = NULL
)
```

Arguments

`bioregionalization`	A `bioregion.clusters` object.
`eval_metrics`	A `character` vector or a single `character` string indicating the metric(s) to be calculated. Available options are `"prop_between_dissim"`, `"anosim"`, `"mean_endemics"` or `"tot_endemics"`. Use `"all"` to compute all available metrics. See Details for metric descriptions.
`dissimilarity`	A site-by-site dissimilarity object from `dissimilarity()` or `dissimilarity_to_similarity()`. Required only for `"prop_between_dissim"` and `"anosim"`.

`dissimilarity_index`
: The name or number of the column to use as dissimilarity. By default, the third column name of `dissimilarity` is used.

`comat`
: A site-species `matrix` with sites as rows and species as columns. Should be provided if `eval_metrics` includes `"avg_endemism"` or `"tot_endemism"`.

`anosim_permutations`
: The number of permutations used to compute the p-value associated with the ANOSIM statistic. Defaults to 1, in which case no p-value is returned.

`eval_metric`
: Deprecated.

`net`
: Deprecated.

`site_col`
: Deprecated.

`species_col`
: Deprecated.

Details

Evaluation metrics:

- **prop_between_dissim**: The proportion of total dissimilarity occurring between bioregions, following Holt et al. (2013), calculated as the sum of between-bioregion dissimilarities divided by the total sum of dissimilarities.
- **anosim**: The Analysis of Similarities (ANOSIM) statistic, based on vegan::anosim(). It measures the separation between within- and between-bioregion dissimilarities. When `permutation` > 1, a p-value is calculated using permutation testing.
- **mean_endemics**: The mean proportion of endemic species across bioregions, following Kreft & Jetz (2010). For each bioregion, the proportion of endemic species is calculated and then averaged across bioregions.
- **tot_endemics**: The proportion of endemic species across all bioregion. It is calculated as the total number of endemic species divided by the total number of species. Endemic species are those occurring in only one bioregion.

Value

A `data.frame` containing the `eval_metrics` values for a bioregionalization and its partition(s).

Author(s)

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

References

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

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

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_2_summary_metrics.html#bioregionalization.

Associated functions: compare_bioregionalizations find_optimal_n

Examples

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

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

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

met <- bioregionalization_metrics(bioreg,
                                  eval_metrics = "all",
                                  dissimilarity = dissim,
                                  comat = comat)
met
```

---

bioregion_colors *Add color palettes to bioregion cluster objects*

Description

This function assigns colors to clusters in a `bioregion.clusters` object using color palettes from the `rcartocolor` package. It handles large numbers of clusters by assigning vivid colors to the most important clusters (based on size), grey shades to less important clusters, and optionally black to insignificant clusters.

Usage

```
bioregion_colors(
  bioregionalization,
  palette = "Vivid",
  cluster_ordering = "n_sites",
  cutoff_insignificant = NULL,
  clusters = NULL
)
```

Arguments

`bioregionalization`
: An object of class `bioregion.clusters`, typically output from clustering functions such as `netclu_infomap()`, `hclu_hierarclust()`, or `nhclu_pam()`.

`palette`
: A `character` string indicating which color palette from `rcartocolor` to use. Default is `"Vivid"`. Other qualitative palettes include `"Bold"`, `"Prism"`, `"Safe"`, `"Antique"`, and `"Pastel"`.

`cluster_ordering`
: A `character` string indicating the criterion for ranking clusters to determine color assignment priority. Options are:

  - `"n_sites"` (default): Rank by number of sites in each cluster
  - `"n_species"`: Rank by number of species (bipartite networks only)
  - `"n_both"`: Rank by combined sites + species (bipartite networks only)

  Larger clusters (by the chosen criterion) receive vivid colors first.

`cutoff_insignificant`
: A `numeric` value or `NULL` (default). When specified, clusters with values at or below this threshold (based on the `cluster_ordering` criterion) are considered insignificant and colored black, reducing visual clutter on maps. If `NULL`, all clusters receive distinct colors.

`clusters`
: Deprecated. Use `bioregionalization` instead.

Details

The function uses a two-step algorithm to assign colors:

Step 1: Identify insignificant clusters (if `cutoff_insignificant` is specified)

Insignificant clusters are those with a marginal size compared to others. This is a subjective threshold set by the user. All such clusters are assigned the color black (#000000) to minimize their visual impact. Clusters with values at or below the threshold are assigned black (#000000).

Step 2: Assign colors to significant clusters

Remaining clusters are ranked by the `cluster_ordering` criterion:

- **Top clusters** (up to 11 for qualitative CARTO palettes, or the palette maximum for other palettes): Receive distinct colors from the chosen palette. For the qualitative palettes (`"Vivid"`, `"Bold"`, `"Prism"`, `"Safe"`, `"Antique"`, `"Pastel"`), the 12th color is always a grey/neutral tone and is automatically excluded.
- **Remaining clusters**: Receive shades of grey from light (#CCCCCC) to dark (#404040), maintaining visual distinction but with less prominence.

Multiple partitions: If the cluster object contains multiple partitions (e.g., from hierarchical clustering with different k values), colors are assigned independently for each partition. Each partition gets its own color scale optimized for the number of clusters in that partition.

Value

A modified `bioregion.clusters` object with two additional elements:

- `colors`: A `list` where each element corresponds to a partition (bioregionalization). Each list element is a `data.frame` with two columns:
  - `cluster` (`character`): Cluster identifier for that partition
  - `color` (`character`): Hex color code (e.g., "#FF5733")
- `clusters_colors`: A `data.frame` with the same structure as the `clusters` element, but with cluster IDs replaced by their corresponding hex color codes for direct use in plotting functions.

Note

The colored cluster object can be directly used with `map_bioregions()`, which will automatically detect and apply the color scheme when present.

Author(s)

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

References

Color palettes from the `rcartocolor` package: Nowosad J (2018). "CARTOColors: color palettes inspired by CARTO." https://github.com/Nowosad/rcartocolor

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_1_visualization.html.

Associated functions: map_bioregions

Examples

```
data(fishmat)
data(fishsf)

# Basic example with few clusters
sim <- similarity(fishmat, metric = "Simpson")
clust <- netclu_greedy(sim)
clust_colored <- bioregion_colors(clust)
print(clust_colored)

## Not run:
# Map with automatic colors
map_bioregions(clust_colored, fishsf)

# Example with many clusters and cutoff
dissim <- similarity_to_dissimilarity(sim)
clust <- hclu_hierarclust(dissim,
```

```
                          optimal_tree_method = "best",
                          n_clust = 15)
clust_colored2 <- bioregion_colors(clust,
                                   cluster_ordering = "n_sites",
                                   cutoff_insignificant = 1)
map_bioregions(clust_colored2, fishsf)

# Example with different palette
clust_colored3 <- bioregion_colors(clust, palette = "Bold")
map_bioregions(clust_colored3, fishsf)


# Example with bipartite network
clust_bip <- netclu_greedy(fishdf, bipartite = TRUE)
clust_bip_colored <- bioregion_colors(clust_bip,
                                      cluster_ordering = "n_both")
map_bioregions(clust_bip_colored, fishsf)


## End(Not run)
```

`bioregion_metrics` *Calculate metrics for bioregions*

Description

This function calculates the number of sites, the number of species, the number of endemic species and the proportion of endemism per bioregion. The spatial coherence can be optionally computed if a spatial object is provided.

Usage

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

Arguments

`bioregionalization`	A `bioregion.clusters` object.
`comat`	A site-species `matrix` with sites as rows and species as columns.
`map`	A spatial object that can be handled by `sf` or `terra`. The first attribute or layer should correspond to the sites' ID (see Details). Needed only for the spatial coherence (`NULL` by default).
`col_bioregion`	Deprecated.

Details

`map` should be the output of `map_bioregions(bioregionalization, geometry, write_clusters = TRUE)`

Value

A `data.frame` with 5 columns (Bioregion ID and metrics described below) or 7 if spatial coherence is computed.

- **n_sites**: Number of sites per bioregion
- **richness**: Number of distinct species per bioregion.
- **rich_endemics**: Number of species found only in the bioregion.
- **prop_endemics**: Fraction of endemics species.
- **sc_size**: Spatial coherence based on size, fraction of the number of site contained in the bioregion's largest contiguous patch.
- **sc_area**: Spatial coherence based on area, fraction of the bioregion area contained in its largest contiguous patch.

Note that if `bioregionalization` contains multiple partitions (i.e., if `dim(bioregionalization$clusters) > 2`), a list will be returned.

Author(s)

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

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_2_summary_metrics.html#bioregion-metrics-spatial-coherence.

Associated functions: site_species_metrics bioregionalization_metrics

Examples

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

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

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

compare_bioregionalizations *Compare cluster memberships among multiple bioregionalizations*

Description

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

Usage

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

Arguments

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

Details

This function operates in two main steps:

1. Within each bioregionalization, the function compares all pairs of items and documents whether they are clustered together (`TRUE`) or separately (`FALSE`). For example, if site 1 and site 2 are clustered in the same cluster in bioregionalization 1, their pairwise membership `site1_site2` will be `TRUE`. This output is stored in the `pairwise_membership` slot if `store_pairwise_membership = TRUE`.
2. Across all bioregionalizations, the function compares their pairwise memberships to determine similarity. For each pair of bioregionalizations, it computes a confusion matrix with the following elements:

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

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

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

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

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

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

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

Value

A `list` containing 4 to 7 elements:

1. **args**: A `list` of user-provided arguments.
2. **inputs**: A `list` containing information on the input bioregionalizations, such as the number of items clustered.
3. **pairwise_membership** (optional): If `store_pairwise_membership = TRUE`, a `boolean matrix` where `TRUE` indicates two items are in the same cluster, and `FALSE` indicates they are not.
4. **freq_item_pw_membership**: A `numeric vector` containing the number of times each item pair is clustered together, corresponding to the sum of rows in `pairwise_membership`.
5. **bioregionalization_freq_cor** (optional): If `cor_frequency = TRUE`, a `numeric vector` of correlations between individual bioregionalizations and the total frequency of pairwise membership.

6. **confusion_matrix** (optional): If `store_confusion_matrix = TRUE`, a `list` of confusion matrices for each pair of bioregionalizations.
7. **bioregionalization_comparison**: A `data.frame` containing comparison results, where the first column indicates the bioregionalizations compared, and the remaining columns contain the requested `indices`.

Author(s)

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

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_3_compare_bioregionalizations.html.

Associated functions: bioregionalization_metrics

Examples

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

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

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

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

# Running the function
compare_bioregionalizations(compare_df)

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

cut_tree *Cut a hierarchical tree*

Description

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

Usage

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

Arguments

`tree`	A `bioregion.hierar.tree` or an `hclust` object.
`n_clust`	An `integer` vector or a single `integer` indicating the number of clusters to be obtained from the hierarchical tree, or the output from `bioregionalization_metrics()`. This should not be used concurrently with `cut_height`.
`cut_height`	A `numeric` vector specifying the height(s) at which the tree should be cut. This should not be used concurrently with `n_clust` or `optim_method`.
`find_h`	A `boolean` indicating whether the cutting height should be determined for the requested `n_clust`.
`h_max`	A `numeric` value indicating the maximum possible tree height for determining the cutting height when `find_h = TRUE`.
`h_min`	A `numeric` value specifying the minimum possible height in the tree for determining the cutting height when `find_h = TRUE`.
`dynamic_tree_cut`	A `boolean` indicating whether the dynamic tree cut method should be used. If `TRUE`, `n_clust` and `cut_height` are ignored.

dynamic_method	A character string specifying the method to be used for dynamically cutting the tree: either "tree" (clusters searched only within the tree) or "hybrid" (clusters searched in both the tree and the dissimilarity matrix).
dynamic_minClusterSize	An integer indicating the minimum cluster size for the dynamic tree cut method (see dynamicTreeCut::cutreeDynamic()).
dissimilarity	Relevant only if dynamic_method = "hybrid". Provide the dissimilarity data.frame used to build the tree.
show_hierarchy	A boolean specifying if the hierarchy of clusters should be identifiable in the outputs (FALSE by default).
verbose	A boolean indicating whether to display progress messages. Set to FALSE to suppress these messages.
...	Additional arguments passed to dynamicTreeCut::cutreeDynamic() to customize the dynamic tree cut method.

Details

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

The dynamic tree cut method has two variants:

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

Value

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

Note

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

Author(s)

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

References

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

See Also

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

Associated functions: hclu_hierarclust

Examples

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

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

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

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

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

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

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

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

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

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

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

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

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

Euclidean computes the Euclidean distance between each pair of sites.

Value

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

Author(s)

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

References

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

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

See Also

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

Associated functions: similarity dissimilarity_to_similarity

Examples

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

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

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

`dissimilarity_to_similarity`

Convert dissimilarity metrics to similarity metrics

Description

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

Usage

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

Arguments

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

Value

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

Note

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

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

Euclidean similarity = 1 / (1 - Euclidean distance)

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

similarity = 1 - dissimilarity

Author(s)

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

See Also

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

Associated functions: similarity dissimilarity_to_similarity

Examples

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

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

similarity <- dissimilarity_to_similarity(dissimil)
similarity
```

exportGDF *Export a network to GDF format for Gephi visualization*

Description

This function exports a network (unipartite or bipartite) from a `data.frame` to the GDF (Graph Data Format) file format, which can be directly imported into Gephi visualization software. The function handles edge data, node attributes, and color specifications.

Usage

```
exportGDF(
  df,
  col1 = "Node1",
  col2 = "Node2",
  weight = NULL,
  bioregions = NULL,
  bioregionalization = NULL,
  color_column = NULL,
  file = "output.gdf"
)
```

Arguments

`df`	A two- or three-column `data.frame` where each row represents an edge (interaction) between two nodes. The first two columns contain the node identifiers, and an optional third column can contain edge weights.
`col1`	A `character` string specifying the name of the first column in `df` containing node identifiers. Defaults to `"Node1"`.
`col2`	A `character` string specifying the name of the second column in `df` containing node identifiers. Defaults to `"Node2"`.
`weight`	A `character` string specifying the name of the column in `df` containing edge weights. If `NULL` (default), edges are unweighted.
`bioregions`	An optional `bioregion.clusters` object (typically from clustering functions like `netclu_greedy()`) or a `data.frame` containing bioregionalization results. When a `bioregion.clusters` object with colors (from `bioregion_colors()`) is provided, colors and bioregion assignments are automatically extracted and used for visualization. Alternatively, a `data.frame` with bioregionalization data can be provided, where each row represents a node with one column containing node identifiers that match those in `df`.
`bioregionalization`	A `character` string or a positive `integer` with two different uses depending on the type of `bioregions`: • When `bioregions` is a `bioregion.clusters` object with multiple partitions: specifies which partition to use. Can be either a character string with

the partition name (e.g., "K_3", "K_5") or a positive integer indicating the partition index (e.g., 1 for first partition, 2 for second). If NULL (default), the first partition is used.

- When bioregions is a data.frame: specifies the name of the column containing node identifiers that match those in df. Must be a character string. Defaults to the first column name if not specified.

color_column A character string specifying the name of a column in bioregions containing color information in hexadecimal format (e.g., "#FF5733"). If specified, colors will be converted to RGB format for Gephi. If NULL (default), colors are automatically extracted when bioregions is a bioregion.clusters object with colors. When bioregions is a plain data.frame, this parameter must be specified to include colors.

file A character string specifying the output file path. Defaults to "output.gdf".

Details

The GDF format is a simple text-based format used by Gephi to define graph structure. This function creates a GDF file with two main sections:

- **nodedef**: Defines nodes and their attributes (name, label, and any additional bioregionalization information from bioregions)
- **edgedef**: Defines edges between nodes, optionally with weights

If color_column is specified, hexadecimal color codes are automatically converted to RGB format (e.g., "#FF5733" becomes "255,87,51") as required by Gephi's color specification.

Attributes are automatically typed as VARCHAR (text), DOUBLE (numeric), or color (for color attributes).

Important note on zero-weight edges: Gephi does not handle edges with weight = 0 properly. If a weight column is specified and edges with weight = 0 are detected, they will be automatically removed from the exported network, and a warning will be issued.

Value

The function writes a GDF file to the specified path and returns nothing (NULL invisibly). The file can be directly opened in Gephi for network visualization and analysis.

Author(s)

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

Examples

```
# Create a simple network
net <- data.frame(
  Node1 = c("A", "A", "B", "C"),
  Node2 = c("B", "C", "C", "D"),
  Weight = c(1.5, 2.0, 1.0, 3.5)
```

```
)

# Export network with weights
## Not run:
exportGDF(net, weight = "Weight", file = "my_network.gdf")

## End(Not run)

# Create bioregionalization data with colors (as data.frame)
bioregion_data <- data.frame(
  node_id = c("A", "B", "C", "D"),
  cluster = c("1", "2", "3", "4"),
  node_color = c("#FF5733", "#33FF57", "#3357FF", "#FF33F5")
)

# Export network with bioregionalization and colors
## Not run:
exportGDF(net,
          weight = "Weight",
          bioregions = bioregion_data,
          bioregionalization = "node_id",
          color_column = "node_color",
          file = "my_network_with_bioregions.gdf")

## End(Not run)

# Using bioregion.clusters object with colors (recommended)
## Not run:
data(fishmat)
net <- similarity(fishmat, metric = "Simpson")
clust <- netclu_greedy(net)
clust_colored <- bioregion_colors(clust)

# Convert to network format
net_df <- mat_to_net(fishmat, weight = TRUE)

# Export with automatic colors from clustering - very simple!
exportGDF(net_df,
          weight = "weight",
          bioregions = clust_colored,
          file = "my_network_colored.gdf")

# With multiple partitions, specify which one to use
dissim <- similarity_to_dissimilarity(similarity(fishmat, metric = "Simpson"))
clust_hier <- hclu_hierarclust(dissim, n_clust = c(3, 5, 8))
clust_hier_colored <- bioregion_colors(clust_hier)

# Using partition name
exportGDF(net_df,
          weight = "weight",
          bioregions = clust_hier_colored,
          bioregionalization = "K_5",
          file = "my_network_K5.gdf")
```

```
# Or using partition index (2 = second partition)
exportGDF(net_df,
          weight = "weight",
          bioregions = clust_hier_colored,
          bioregionalization = 2,
          file = "my_network_partition2.gdf")

## End(Not run)
```

`find_optimal_n` *Search for an optimal number of bioregions in a bioregionalization*

Description

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

Usage

```
find_optimal_n(
  evaluation_df,
  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,
  verbose = TRUE,
  bioregionalizations = NULL
)
```

Arguments

`evaluation_df`	a `data.frame` output from `bioregionalization_metrics()`) or a `data.frame` with the first two columns with partition name and number of bioregions, followed by columns with numeric evaluation metrics.
`metrics_to_use`	A `character` vector or single string specifying metrics in `evaluation_df` for calculating optimal bioregions. Defaults to `"all"` (uses all metrics).
`criterion`	A `character` string specifying the criterion to identify optimal clusters. Options include `"elbow"`, `"increasing_step"`, `"decreasing_step"`, `"cutoff"`, `"breakpoints"`, `"min"`, or `"max"`. Defaults to `"elbow"`. See Details.

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

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

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

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

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

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

verbose A boolean indicating whether to display progress messages. Set to FALSE to suppress these messages.

bioregionalizations Deprecated.

Details

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

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

Criteria for optimal clusters:

- elbow: Identifies the "elbow" point in the evaluation metric curve, where incremental improvements diminish. Based on a method to find the maximum distance from a straight line linking curve endpoints.
- increasing_step or decreasing_step: Highlights significant increases or decreases in metrics by analyzing pairwise differences between bioregionalizations. Users specify step_quantile or step_levels.
- cutoffs: Derives clusters from specified metric cutoffs, e.g., as in Holt et al. (2013). Adjust cutoffs based on spatial scale.
- breakpoints: Uses segmented regression to find breakpoints. Requires specifying n_breakpoints.
- min & max: Selects clusters at minimum or maximum metric values.

Value

A listcontaining these elements:

- args: Input arguments.
- evaluation_df: The input evaluation data.frame, appended with boolean columns for optimal cluster counts.

- optimal_n: A list with optimal cluster counts for each metric in "metrics_to_use", based on the chosen criterion.
- plot: The plot (if requested).

Note

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

Author(s)

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

References

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

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

See Also

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

Associated functions: hclu_hierarclust

Examples

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

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

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

evalmet <- bioregionalization_metrics(bioreg,
                                      eval_metrics = "anosim",
```

```
                                    dissimilarity = dissim)

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

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

Description

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

Usage

```
fishdf
```

Format

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

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

Species Unique species identifier

Abundance Species abundance

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

Description

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

Usage

```
fishmat
```

Format

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

fishsf *Spatial distribution of fish in Europe*

Description

A dataset containing the geometry of the 338 sites.

Usage

```
fishsf
```

Format

A `sf` `data.frame` with 338 rows and 2 columns:

Site Unique site identifier

geometry Geometry of the site

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

Description

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

Usage

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

Arguments

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

Details

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

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

Value

A list of class bioregion.clusters with five slots:

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

Author(s)

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

References

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

See Also

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

Associated functions: cut_tree

Examples

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

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

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

---

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

Description

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

Usage

```
hclu_hierarclust(
  dissimilarity,
  index = names(dissimilarity)[3],
  method = "average",
  randomize = TRUE,
  seed = NULL,
  n_runs = 100,
  keep_trials = "no",
  optimal_tree_method = "ihct",
```

```
  n_clust = NULL,
  cut_height = NULL,
  find_h = TRUE,
  h_max = 1,
  h_min = 0,
  consensus_p = 0.5,
  show_hierarchy = FALSE,
  ihct_top_n_trees = 2,
  ihct_variation_drop = 0.2,
  ihct_sites_drop = 10,
  ihct_height_rule = "least_squares",
  ihct_n_workers = 1,
  verbose = TRUE
)
```

Arguments

dissimilarity	The output object from dissimilarity() or similarity_to_dissimilarity(), or a dist object. If a data.frame is used, the first two columns represent pairs of sites (or any pair of nodes), and the subsequent column(s) contain the dissimilarity indices.
index	The name or number of the dissimilarity column to use. By default, the third column name of dissimilarity is used.
method	The name of the hierarchical classification method, as in hclust. Should be one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC), or "centroid" (= UPGMC).
randomize	A boolean indicating whether the dissimilarity matrix should be randomized to account for the order of sites in the dissimilarity matrix.
seed	A value for the random number generator (NULL for random by default).
n_runs	The number of trials for randomizing the dissimilarity matrix.
keep_trials	A character string indicating whether random trial results (including the randomized matrix, the associated tree and metrics for that tree) should be stored in the output object. Possible values are "no" (default), "all" or "metrics". Note that this parameter is automatically set to "no" if optimal_tree_method = "ihct".
optimal_tree_method	A character string indicating how the final tree should be obtained from all trials. Possible values are "ihct" (default), "best" or "consensus". "iterative_consensus_tree" is still accepted as another name for "ihct", so that code written for earlier versions of bioregion keeps working. **We recommend "ihct". See Details.**
n_clust	An integer vector or a single integer indicating the number of clusters to be obtained from the hierarchical tree, or the output from bioregionalization_metrics. This parameter should not be used simultaneously with cut_height.
cut_height	A numeric vector indicating the height(s) at which the tree should be cut. This parameter should not be used simultaneously with n_clust.

find_h A boolean indicating whether the height of the cut should be found for the requested n_clust.

h_max A numeric value indicating the maximum possible tree height for the chosen index.

h_min A numeric value indicating the minimum possible height in the tree for the chosen index.

consensus_p A numeric value (applicable only if optimal_tree_method = "consensus") indicating the threshold proportion of trees that must support a region/cluster for it to be included in the final consensus tree.

show_hierarchy A boolean specifying if the hierarchy of clusters should be identifiable in the outputs (FALSE by default). This argument is only used if the tree is cut (i.e., n_clust or cut_height is provided).

ihct_top_n_trees

An integer (applicable only if optimal_tree_method = "ihct") indicating how many of the best randomized trees are used to decide each division of the tree (2 by default). See Details.

ihct_variation_drop

A numeric value between 0 and 1 (applicable only if optimal_tree_method = "ihct" and method = "average"). This is an optimization parameter: it reduces the number of times the dissimilarity matrix is randomized while the tree is built, which makes the tree faster to obtain for a very small cost in its fit to the data. Its rule is based on variation: new randomizations are made once a group of sites has lost this share of the variation that was left to decide when they were last made. The default 0.2 therefore means "randomize again once a fifth of what was left to decide has been decided". Set it to 0 to randomize at every division, or to 1 to switch this rule off and leave ihct_sites_drop to decide on its own. See Details.

ihct_sites_drop

A numeric value of 0 or more (applicable only if optimal_tree_method = "ihct", method = "average" and ihct_variation_drop > 0). This is a second optimization parameter, used together with ihct_variation_drop, with a rule based on sites rather than on variation: new randomizations are also made once a group of sites has lost this many sites since they were last made (10 by default). Set it to 0 or 1 to randomize at every division, or to Inf to switch this rule off and leave ihct_variation_drop to decide on its own. See Details.

ihct_height_rule

A character string (applicable only if optimal_tree_method = "ihct") indicating how the heights of the tree are corrected when a division comes out lower than a division it contains. With "least_squares" (default) the heights are moved as little as possible, which fits the dissimilarities better; with "max_child" each division is raised to the highest division it contains, as in bioregion 1.4.0 and earlier. See Details.

ihct_n_workers An integer of 1 or more (applicable only if optimal_tree_method = "ihct") indicating how many processes of your computer may build the randomized trees at the same time. With 1 (default) they are built one after another, as before. Higher values are worth it on large matrices only, and they do not change the tree: the same seed gives the same result whatever this is set to. See Details.

verbose A boolean indicating whether to display progress messages. Set to FALSE to suppress these messages.

Details

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

Clusters can be obtained by two methods:

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

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

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

Two methods are available to obtain the final tree:

- optimal_tree_method = "ihct": The Iterative Hierarchical Consensus Tree (IHCT) method reconstructs a consensus tree by iteratively splitting the dataset into two subclusters based on the pairwise dissimilarity of sites across n_runs trees based on n_runs randomizations of the distance matrix. At each iteration, it identifies the majority membership of sites into two stable groups across all trees, calculates the height based on the selected linkage method (method), and enforces monotonic constraints on node heights to produce a coherent tree structure. This approach provides a robust, hierarchical representation of site relationships, balancing cluster stability and hierarchical constraints.
- optimal_tree_method = "best": This method selects one tree among with the highest cophenetic correlation coefficient, representing the best fit between the hierarchical structure and the original distance matrix.
- optimal_tree_method = "consensus": This method constructs a consensus tree using phylogenetic methods with the function consensus. When using this option, you must set the consensus_p parameter, which indicates the proportion of trees that must contain a region/cluster for it to be included in the final consensus tree. Consensus trees lack an inherent height because they represent a majority structure rather than an actual hierarchical clustering. To assign heights, we use a non-negative least squares method (nnls.tree) based on the initial distance matrix, ensuring that the consensus tree preserves approximate distances among clusters.

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

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

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

~ Iterative Hierarchical Consensus Tree details ~

The paragraphs below cover the `ihct_*` arguments of this function. The algorithm itself is walked through step by step in `ihct()`, which is the function doing the work here, and which can also be called on its own if you want the tree without the rest of this function.

–> Tree quality

`ihct_top_n_trees` sets how many of the randomized trees, ranked by their quality (i.e., how well they fit the dissimilarities with cophenetic correlation coefficient CCC), are used to decide a division: with `1` the division is the top division of the best tree; with more, sites are grouped according to how often they fall on the same side in these trees. We recommend leaving this to default values, or to a low number of trees, because it provided the best results in our tests (highest CCC).

–> Computation time

The algorithm can be long to run because at every division of the tree it has to randomize `n_runs` trees. On large datasets, building new trees at every division makes this method slow. To make the function usable on large datasets, we provide two optimization parameters. These parameters make the algorithm reuse previously randomized trees at new division, unless a threshold of change is reached:

- `ihct_variation_drop` is based on the amount of variation from the dissimilarity matrix. It triggers a new tree randomization only when the amount of variation in the group being divided has reached a threshold since last randomization (default: 20% drop in variation). In other words, new randomizations happen only when tree divisions reach a certain threshold of variation since the last randomization. For example, when a tree peels off only 1 site at a time, this argument makes sure no new randomization trigger unless variability reaches the desired threshold.
- `ihct_sites_drop`is based on how many sites the group has lost. It triggers new randomizations only when a certain number of sites have been excluded since the last randomization.

These two parameters with their defaults (`ihct_variation_drop = 0.2`, `ihct_sites_drop = 10`) result in marginal changes in algorithm performance (loss in CCC <0.001) and make the tree 1.5 faster to build. In our tests, the larger the datasets, the higher the savings with these two optimization parameters. Note, however, that reusing trees only works with UPGMA currently, so it only applies to `method = "average"`.

The two arguments work as a pair, and each of them can be set so that the other no longer has any effect. A new randomization is made as soon as *either* of them asks for one, so whichever of the two asks more often is the one that decides:

- `ihct_sites_drop = 1` (or `0`) means new randomizations every time a site is treated (so randomizations at every division, and `ihct_variation_drop` is never used).
- `ihct_variation_drop = 0` likewise means new randomizations at every division, and `ihct_sites_drop` is then never used.
- `ihct_variation_drop = 1` never triggers new randomizations, leaving `ihct_sites_drop` to decide on its own, and `ihct_sites_drop = Inf` never triggers randomizations, leaving `ihct_variation_drop` to decide on its own.
- Both switched off (`ihct_variation_drop = 1` and `ihct_sites_drop = Inf`) randomizes once, at the first division, and reuses those trees for the whole tree. This is the fastest setting and the one that fits the data least well.

To reproduce the tree that bioregion 1.4.0 and earlier produced, for the same seed, use:

```
hclu_hierarclust(dissimilarity,
                 method = "average",
                 optimal_tree_method = "ihct",
                 n_runs = 100,
                 ihct_top_n_trees = 2,
                 ihct_variation_drop = 0,
                 ihct_height_rule = "max_child")
```

–> Height of nodes in the tree

The height of nodes in a tree must be monotonous, i.e. a child node cannot be have a higher height than its parents. However, this situation can happen when building the tree, which is why all tree construction algorithms have a monotonicity section where node height is recalculated.

`ihct_height_rule` decides how we do it in IHCT. `"max_child"` raises every division to the highest division it contains, which is simple but can push a division far above the dissimilarities it summarizes. `"least_squares"` (default) instead moves the heights as little as possible, which with `method = "average"` gives the heights that fit the dissimilarities best on the topology at hand, so the cophenetic correlation is never below the one `"max_child"` gives and is usually above it. We recommend leaving to default as in our own testing it provides better performance (highest CCC).

–> Parallelization for quicker computation time

Most of the waiting is spent building the randomized trees, and the runs of one group do not depend on each other, so `ihct_n_workers` can share them between several processes of your computer. This only pays on large matrices, where a single run is slow enough to be worth sending to another process: groups of fewer than 200 sites are always done in one process, and small datasets should be left at `ihct_n_workers = 1`. We found that 4 workers give good gains (about three times faster on a 5,000-site matrix); beyond that the processes spend their time waiting for memory rather than computing, and on a 10,000-site matrix going from 4 workers to 8 provided only limited gains while doubling the memory needed. Each worker also needs its own copy of the dissimilarity matrix on Windows, about 200 MB for 5,000 sites and 800 MB for 10,000, so ask for fewer workers than your memory allows copies. Whatever you set, the tree is the same: the random shuffles are always drawn in the same order by the main process, and only the building of the trees is handed out to workers.

Value

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

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

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

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

Author(s)

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

References

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

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

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

See Also

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

Associated functions: cut_tree ihct

Examples

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

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

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

# User-defined height cut
# Only one height
tree2 <- hclu_hierarclust(dissim,
```

```
                          cut_height = .05)
tree2
tree2$clusters

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

tree3$clusters # Mind the order of height cuts: from deep to shallow cuts
# Info on each partition can be found in table cluster_info
tree3$cluster_info
plot(tree3)
```

`hclu_optics` *OPTICS hierarchical clustering algorithm*

Description

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

Usage

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

Arguments

`dissimilarity`	The output object from `dissimilarity()` or `similarity_to_dissimilarity()`, or a `dist` object. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the subsequent column(s) contain the dissimilarity indices.
`index`	The name or number of the dissimilarity column to use. By default, the third column name of `dissimilarity` is used.
`minPts`	A `numeric` value specifying the minPts argument of dbscan. minPts is the minimum number of points required to form a dense region. By default, it is set to the natural logarithm of the number of sites in `dissimilarity`, rounded down to a whole number.

eps	A positive numeric value specifying the eps argument of optics. It defines the upper limit of the size of the epsilon neighborhood. Limiting the neighborhood size improves performance and has no or very little impact on the ordering as long as it is not set too low. By default, eps = Inf, which means that the neighborhood size is not limited.
xi	A numeric value specifying the steepness threshold to identify clusters hierarchically using the Xi method (see optics).
minimum	A boolean specifying whether the hierarchy should be pruned from the output to only retain clusters at the "minimal" level, i.e., only leaf / non-overlapping clusters. If TRUE, then the argument show_hierarchy should be set to FALSE.
show_hierarchy	A boolean specifying whether the hierarchy of clusters should be included in the output. By default, the hierarchy is not visible in the clusters obtained from OPTICS; it can only be visualized by plotting the OPTICS object. If show_hierarchy = TRUE, the output cluster data.frame will contain additional columns showing the hierarchy of clusters.
algorithm_in_output	A boolean indicating whether the original output of dbscan should be returned in the output (TRUE by default, see Value).
...	Additional arguments to be passed to optics() (see optics).

Details

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

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

Value

A list of class bioregion.clusters with five slots:

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

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

Author(s)

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

References

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

See Also

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

Associated functions: nhclu_dbscan

Examples

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

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

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

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

---

ihct *Iterative hierarchical consensus tree from a dissimilarity matrix*

Description

This function builds a hierarchical tree of sites from a matrix of pairwise dissimilarities, using the Iterative Hierarchical Consensus Tree (IHCT) algorithm. Unlike an ordinary hierarchical clustering, the tree does not depend on the order in which the sites happen to be stored in the matrix.

Most users should use `hclu_hierarclust()` and not this function. `ihct()` is an algorithm used in `hclu_hierarclust()`, exported for users who want the algorithm itself for their purposes.

Usage

```
ihct(
  dist_mat,
  method = "average",
  n_runs = 100,
  top_n_trees = 2,
  variation_drop = 0.2,
  sites_drop = 10,
  height_rule = c("least_squares", "max_child"),
  tie_block_resolution = TRUE,
  n_workers = 1,
  size_parallel = 200,
  verbose = TRUE
)
```

Arguments

`dist_mat`	A square `matrix` of dissimilarities between sites, with the site names as row and column names. A `dist` object is also accepted and converted.
`method`	The name of the linkage method, as in hclust. It is used both to build the randomized trees and to give every division of the final tree its height. Should be one of `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"` (= UPGMA, the default), `"mcquitty"` (= WPGMA), `"median"` (= WPGMC) or `"centroid"` (= UPGMC). Two of the optimization parameters described in details are only available with some of these.
`n_runs`	The number of randomized trees built to decide one division (`100` by default). More trees make each division more stable and the whole tree slower to obtain.
`top_n_trees`	An `integer` indicating how many of the randomized trees, ranked by how well they fit the dissimilarities, decide a division (`2` by default). With `1` the division is read straight off the best tree; with more, sites are grouped according to how often they end up on the same side across those trees. See details.
`variation_drop`	A `numeric` value between 0 and 1 (`0.2` by default). This is a speed optimization parameter: it lets a group of sites reuse the randomized trees of the group it was split from instead of building its own, which is faster and costs a very small amount of fit. Its rule is based on variation: fresh trees are built once a group has lost this share of the variation it still held when its current trees were built. Set it to `0` to build fresh trees at every division, or to `1` to switch this rule off and leave `sites_drop` in charge. Only used with `method = "average"`. See Details.
`sites_drop`	A `numeric` value of 0 or more (`10` by default). This is the second speed optimization parameter, working alongside `variation_drop`, with a rule based on sites rather than on variation: fresh trees are also built once a group has lost this many sites since its current trees were built. Set it to `0` or `1` to build fresh trees at every division, or to `Inf` to switch this rule off and leave `variation_drop` in charge. Only used with `method = "average"` and `variation_drop > 0`. See Details.
`height_rule`	A `character` string indicating how the heights of the tree are corrected when a division comes out lower than a division it contains. With `"least_squares"`

(default) the heights are moved as little as possible, which fits the dissimilarities better; with `"max_child"` every division is raised to the highest division it contains, as in bioregion 1.4.0 and earlier. See Details.

`tie_block_resolution`
: A `boolean` (`TRUE` by default) deciding what happens to a group of sites whose dissimilarities are all equal, where the randomization has nothing left to choose between. With `TRUE` such a group is resolved directly at that value, which saves its `n_runs` trees and costs nothing in fit; with `FALSE` it is divided from randomized trees like any other group. Only used with `method` set to `"single"`, `"complete"`, `"average"` or `"mcquitty"`. See Details.

`n_workers`
: An `integer` of 1 or more indicating how many processes of your computer build the randomized trees at the same time. With `1` (default) they are built one after another. Higher values are worth it on large matrices only, and they do not change the tree: the same seed gives the same result whatever this is set to. See Details.

`size_parallel`
: An `integer` of 2 or more indicating the smallest group of sites whose randomized trees are worth handing out to the worker processes (`200` by default). Groups smaller than this are always done in one process, because sending them out costs more than it saves. Only used when `n_workers > 1`.

`verbose`
: A `boolean` indicating whether to display progress and information messages. Set to `FALSE` to suppress them.

Details

— 1. The problem: the order of the sites changes the tree —:

A dissimilarity matrix computed from species composition contains a great many identical values, because many pairs of sites share exactly the same number of species. Hierarchical clustering has to break those ties somehow, and hclust breaks them by taking whichever pair comes first in the matrix. Shuffle the rows of the matrix and you get a different tree out of the very same data (Dapporto et al., 2013). IHCT removes that arbitrariness by generating a true consensus tree from many randomizations.

— 2. The tree is built from the top down —:

IHCT does not join sites into ever larger groups, the way ordinary hierarchical clustering does. It starts with a single group containing every site, cuts that group in two, then cuts each of the two in two, and carries on until every group holds a single site. Each of those cuts is called a division, and the whole algorithm is about deciding one division well. Divisions are always strictly binary (i.e., one branch is always cut into two branches).

— 3. How one division is decided —:

Take a group of sites that has to be divided. Four things happen to it:

1. Its sites are put in a random order and a tree is built on them, with hclust and the linkage of `method`. This is repeated `n_runs` times, giving `n_runs` trees that differ only in how the ties were broken.
2. Each of those trees is scored by its cophenetic correlation coefficient, that is, by how closely the distances read off the tree reproduce the dissimilarities the tree was built from. The `top_n_trees` best-scoring trees are kept and the rest are discarded.

3. From the kept trees, we take the top division to decide how to split the group of sites into two. With `top_n_trees = 1` we take the top division of the best tree. With more, we take a consensus division based on how often sites are on the same side across the kept trees. The rule we used is that the consensus is based on the majority decision, sites are grouped together if theyr are together in > 50% of trees.
4. The division is given a height, computed from the dissimilarities between the two groups it separates in the way `method` prescribes: their mean for `"average"` (UPGMA), their smallest value for `"single"`, their largest for `"complete"`, and so on.

For each of the two groups that come out of the division, the algorithm run them through the same four steps, and so on for their own halves, until we reach single sites.

This why the algorithm is called "iterative": it randomizes trees again at every division from top to bottom. It is why this method provides better performance compareds to other approaches: a tie broken one way at the top of the tree does not force the same choice at the divisions underneath it.

Since the randomization uses R's random number generator, call `set.seed()` before `ihct()` if you want the same tree twice.

— 4. Computaion time and how to shorten it —:

Step 1 above is where nearly all of the computing time goes: `n_runs` trees are built at every single division. However, the trees usually peel a handful of sites off the rest at each division rather than splitting the data in half. This means randomizing everything at every division can be inefficient, resulting in large computation times, making the function too long to use on large datasets.

To make the function usable on large datasets, we provide two optimization parameters. These parameters make the algorithm reuse previously randomized trees at new division, unless a threshold of change is reached:

- `variation_drop` is based on the amount of variation from the dissimilarity matrix. It triggers a new tree randomization only when the amount of variation in the group being divided has reached a threshold since last randomization (default: 20% drop in variation). In other words, new randomizations happen only when tree divisions reach a certain threshold of variation since the last randomization. For example, when a tree peels off only 1 site at a time, this argument makes sure no new randomization trigger unless variability reaches the desired threshold.
- `sites_drop`is based on how many sites the group has lost. It triggers new randomizations only when a certain number of sites have been excluded since the last randomization.

These two parameters with their defaults (`variation_drop = 0.2`, `sites_drop = 10`) result in marginal changes in algorithm performance (loss in CCC <0.001) and make the tree 1.5 faster to build. In our tests, the larger the datasets, the higher the savings with these two optimization parameters. Note, however, that reusing trees only works with UPGMA currently, so it only applies to `method = "average"`.

The two arguments work as a pair, and each of them can be set so that the other no longer has any effect. A new randomization is made as soon as *either* of them asks for one, so whichever of the two asks more often is the one that decides:

- `sites_drop = 1` (or `0`) means new randomizations every time a site is treated (so randomizations at every division, and `variation_drop` is never used).
- `variation_drop = 0` likewise means new randomizations at every division, and `sites_drop` is then never used.

- `variation_drop = 1` never triggers new randomizations, leaving `sites_drop` to decide on its own, and `sites_drop = Inf` never triggers randomizations, leaving `variation_drop` to decide on its own.
- Both switched off (`variation_drop = 1` and `sites_drop = Inf`) randomizes once, at the first division, and reuses those trees for the whole tree. This is the fastest setting and the one that fits the data least well.

— 5. Groups where all distances are equal —:

Some groups have all their dissimilarities equal to one single value: every pair of sites inside the group is exactly as different as every other pair. These are tied blocks, and they are common in presence-absence data, where indices such as Simpson saturate at 1 as soon as two sites share no species. A tied block means randomization brings nothing useful and the `n_runs` trees are wasted computation time.

To avoid this, the argument `tie_block_resolution = TRUE` (the default) recognises such a group and resolves it directly, peeling its sites off one at a time with every division sitting at the common value. Setting it to `FALSE` divides tied blocks from randomized trees like any other group (pre-1.4.0 behaviour).

— 6. Node heights —:

The height of nodes in a tree must be monotonous, i.e. a child node cannot be have a higher height than its parents. However, this situation can happen when building the tree, which is why all tree construction algorithms have a monotonicity section where node height is recalculated.

IHCT corrects the heights once the whole tree is built, with a method that depends on `height_rule`:

`"max_child"` raises every division to the highest division inside it. It is simple, and it is what bioregion 1.4.0 and earlier did, but a single high division buried deep inside a group drags all of its parents up with it, well above the dissimilarities those divisions actually summarize. We found out it provides lower quality (lower cophenetic correlation coefficient) than `"least_squares"`, so we changed the default after 1.4.0.

`"least_squares"` (the default) instead moves the heights as little as it can: among all sets of heights with no branch doubling back, it takes the one that stays closest to the heights the divisions were given, weighting each division by the number of site pairs it stands for. With `method = "average"` these are provably the heights that fit the dissimilarities best on the tree shape at hand, so the cophenetic correlation is never below what `"max_child"` gives and is usually a little above it.

— 7. Computation time and parallelization —:

Most of the waiting is spent building the randomized trees, and the runs of one group do not depend on each other, so `ihct_n_workers` can share them between several processes of your computer. This only pays on large matrices, where a single run is slow enough to be worth sending to another process: groups of fewer than 200 sites are always done in one process, and small datasets should be left at `ihct_n_workers = 1`. We found that 4 workers give good gains (about three times faster on a 5,000-site matrix); beyond that the processes spend their time waiting for memory rather than computing, and on a 10,000-site matrix going from 4 workers to 8 provided only limited gains while doubling the memory needed. Each worker also needs its own copy of the dissimilarity matrix on Windows, about 200 MB for 5,000 sites and 800 MB for 10,000, so ask for fewer workers than your memory allows copies. Whatever you set, the tree is the same: the random shuffles are always drawn in the same order by the main process, and only the building of the trees is handed out to workers.

— 8. Reproducing the trees of bioregion 1.4.0 —:

For the same seed, the following settings give the tree that bioregion 1.4.0 and earlier produced:

```
ihct(dist_mat,
     method = "average",
     n_runs = 100,
     top_n_trees = 2,
     variation_drop = 0,
     height_rule = "max_child",
     tie_block_resolution = FALSE)
```

`sites_drop` may be left at any value here, since `variation_drop = 0` already rebuilds the trees at every division.

Value

An `hclust` object, whose `method` element is set to `"Iterative Hierarchical Consensus Tree"`. Its heights are monotone (no division is lower than a division it contains) and its labels are the site names of `dist_mat`, in alphabetical order. It can be plotted, cut with `cut_tree()` or cutree, and used anywhere an `hclust` object is expected.

Author(s)

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

References

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

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

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

See Also

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

Associated functions: hclu_hierarclust cut_tree

Examples

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

```
dissim <- dissimilarity(comat, metric = "Simpson")
dist_mat <- net_to_mat(dissim[, 1:3],
                       weight = TRUE,
                       squared = TRUE,
                       symmetrical = TRUE)

set.seed(1)
tree <- ihct(dist_mat,
             n_runs = 20,
             verbose = FALSE)
plot(tree)
cut_tree(tree, n_clust = 3)

# Fastest setting: build the randomized trees once, at the first division,
# and reuse them all the way down
set.seed(1)
fast_tree <- ihct(dist_mat,
                  n_runs = 20,
                  variation_drop = 1,
                  sites_drop = Inf,
                  verbose = FALSE)
```

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

Description

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

Usage

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

Arguments

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

infomap_version
: A character vector or a single character string specifying the Infomap version(s) to install.

verbose
: A boolean indicating whether to display progress messages. Set to FALSE to suppress these messages.

Details

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

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

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

Value

No return value.

Note

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

Author(s)

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

See Also

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

---

map_bioregions *Create a map of bioregions*

Description

This plot function can be used to visualize bioregions based on a bioregion.clusters object combined with a spatial object (sf or terra).

Usage

```
map_bioregions(
  bioregionalization,
  map,
  partition_index = NULL,
  map_as_output = FALSE,
  plot = TRUE,
  clusters = NULL,
  geometry = NULL,
  write_clusters = NULL,
  ...
)
```

Arguments

bioregionalization	A bioregion.clusters object.
map	A spatial object that can be handled by sf or terra. The first attribute or layer should correspond to the sites' ID (see Details).
partition_index	An integer, character, or NULL specifying which bioregionalization's partition(s) to plot. By default (NULL), all partitions are plotted. If an integer or vector of integers is provided, partition(s) are selected by column number(s) in the bioregionalization data.frame. If a character or vector of characters, partition(s) are selected by name(s) matching column names in bioregionalization.
map_as_output	A boolean indicating if the sf data.frame object used for the plot should be returned.
plot	A boolean indicating if the plot should be drawn.
clusters	Deprecated. Use bioregionalization instead. The former bioregionalization has been replaced by partition_index.
geometry	Deprecated. Use map instead.
write_clusters	Deprecated. Use map_as_output instead.
...	Further arguments to be passed to sf::plot().

Details

The site IDs in bioregionalization and map should correspond. They must have the same type (i.e., character if bioregionalization is a bioregion.clusters object), and the sites in bioregionalization should be included among the sites in map. If map is an sf or a SpatVector (terra) object, it should contain an attribute table with the IDs in the first column. If map is a SpatRaster (terra) object, it should contain the IDs in the first layer.

If the bioregionalization object contains both types of nodes (sites and species), only site will be mapped. The function automatically filters to site nodes using the node_type attribute.

Colors: If the bioregionalization object contains colors (added via bioregion_colors()), these colors will be automatically used for plotting. Otherwise, the default sf color scheme will be applied.

Value

One or several maps of bioregions if `plot = TRUE` and the `sf data.frame` object used for the plot if `map_as_output = TRUE`.

Author(s)

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

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_1_visualization.html.

Associated functions: bioregion_colors

Examples

```
data(fishmat)
data(fishsf)

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

# With colors
clu_colored <- bioregion_colors(clu)
mapclu <- map_bioregions(clu_colored,
                         map = fishsf,
                         map_as_output = TRUE,
                         plot = FALSE)
```

---

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

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

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

See Also

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

Associated functions: net_to_mat

Examples

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

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

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

Description

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

Usage

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

Arguments

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

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

Details

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

Value

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

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

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

Note

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

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

com <- netclu_beckett(net)
```

`netclu_greedy` *Community structure detection via greedy optimization of modularity*

Description

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

Usage

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

Arguments

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

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

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

Details

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

Value

A list of class bioregion.clusters with five slots:

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

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

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

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

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

---

netclu_infomap *Infomap community finding*

Description

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

Usage

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

```
  check_install = TRUE,
  path_temp = "infomap_temp",
  delete_temp = TRUE
)
```

Arguments

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

Details

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

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

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

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

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

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

Value

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

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

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

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_greedy netclu_louvain netclu_oslom

Examples

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

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

---

netclu_labelprop *Finding communities based on propagating labels*

Description

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

Usage

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

Arguments

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

Details

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

Value

A list of class bioregion.clusters with five slots:

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

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

Note

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

Do not forget to indicate which of the first two columns is dedicated to the site nodes (i.e., primary nodes) and species nodes (i.e. feature nodes) using the arguments site_col and species_col. The

type of nodes returned in the output can be chosen with the argument `return_node_type` equal to `"both"` to keep both types of nodes, `"site"` to preserve only the site nodes, and `"species"` to preserve only the species nodes.

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

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

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

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

---

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

Description

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

Usage

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

Arguments

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

Details

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

Value

A list of class bioregion.clusters with five slots:

1. **name**: A character containing the name of the algorithm.
2. **args**: A list of input arguments as provided by the user.
3. **inputs**: A list of characteristics of the clustering process.

4. **algorithm**: A `list` of all objects associated with the clustering procedure, such as original cluster objects (only if `algorithm_in_output = TRUE`).
5. **clusters**: A `data.frame` containing the clustering results.

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

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

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

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

netclu_leiden *Finding communities using the Leiden algorithm*

Description

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

Usage

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

Arguments

net	The output object from similarity() or dissimilarity_to_similarity(). If a data.frame is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
weight	A boolean indicating if the weights should be considered if there are more than two columns.
cut_weight	A minimal weight value. If weight is TRUE, the links between sites with a weight strictly lower than this value will not be considered (0 by default).
index	The name or number of the column to use as weight. By default, the third column name of net is used.
seed	The random number generator seed (NULL for random by default).
objective_function	A string indicating the objective function to use, either the Constant Potts Model ("CPM") or "modularity" ("CPM" by default).
resolution_parameter	The resolution parameter to use. Higher resolutions lead to smaller communities, while lower resolutions lead to larger communities.

`beta` A parameter affecting the randomness in the Leiden algorithm. This affects only the refinement step of the algorithm.

`n_iterations` The number of iterations for the Leiden algorithm. Each iteration may further improve the partition.

`vertex_weights` The vertex weights used in the Leiden algorithm. If not provided, they will be automatically determined based on the objective_function. Please see the details of this function to understand how to interpret the vertex weights.

`bipartite` A `boolean` indicating if the network is bipartite (see Details).

`site_col` The name or number for the column of site nodes (i.e., primary nodes).

`species_col` The name or number for the column of species nodes (i.e., feature nodes).

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

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

Details

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

Value

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

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

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

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

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

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

---

netclu_louvain *Louvain community finding*

Description

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

Usage

```
netclu_louvain(
  net,
  weight = TRUE,
  cut_weight = 0,
  index = names(net)[3],
  lang = "igraph",
  resolution = 1,
  seed = NULL,
```

```
  q = 0,
  c = 0.5,
  k = 1,
  bipartite = FALSE,
  site_col = 1,
  species_col = 2,
  return_node_type = "both",
  binpath = "tempdir",
  check_install = TRUE,
  path_temp = "louvain_temp",
  delete_temp = TRUE,
  algorithm_in_output = TRUE
)
```

Arguments

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

check_install A boolean indicating if the function should check that Louvain has been properly installed (see install_binaries and Details).

path_temp A character indicating the path to the temporary folder (see Details).

delete_temp A boolean indicating if the temporary folder should be removed (see Details).

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

Details

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

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

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

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

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

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

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

Value

A list of class bioregion.clusters with five slots:

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

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

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_greedy netclu_oslom

Examples

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

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

`netclu_oslom` *OSLOM community finding*

Description

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

Usage

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

Arguments

`net`	The output object from `similarity()` or `dissimilarity_to_similarity()`. If a `data.frame` is used, the first two columns represent pairs of sites (or any pair of nodes), and the next column(s) are the similarity indices.
`weight`	A `boolean` indicating if the weights should be considered if there are more than two columns.
`cut_weight`	A minimal weight value. If `weight` is TRUE, the links between sites with a weight strictly lower than this value will not be considered (0 by default).
`index`	Name or number of the column to use as weight. By default, the third column name of `net` is used.
`seed`	For the random number generator (NULL for random by default).
`reassign`	A `character` indicating if the nodes belonging to several community should be reassigned and what method should be used (see Note).

r	The number of runs for the first hierarchical level (10 by default).
hr	The number of runs for the higher hierarchical level (50 by default, 0 if you are not interested in hierarchies).
t	The p-value, the default value is 0.10. Increase this value if you want more modules.
cp	Kind of resolution parameter used to decide between taking some modules or their union (default value is 0.5; a bigger value leads to bigger clusters).
directed	A boolean indicating if the network is directed (from column 1 to column 2).
bipartite	A boolean indicating if the network is bipartite (see Details).
site_col	Name or number for the column of site nodes (i.e. primary nodes).
species_col	Name or number for the column of species nodes (i.e. feature nodes).
return_node_type	A character indicating what types of nodes (site, species, or both) should be returned in the output (return_node_type = "both" by default).
binpath	A character indicating the path to the bin folder (see install_binaries and Details).
check_install	A boolean indicating if the function should check that the OSLOM has been properly installed (see install_binaries and Details).
path_temp	A character indicating the path to the temporary folder (see Details).
delete_temp	A boolean indicating if the temporary folder should be removed (see Details).

Details

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

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

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

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

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

Value

A list of class bioregion.clusters with five slots:

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

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

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

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_greedy netclu_infomap netclu_louvain

Examples

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

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

`netclu_walktrap` *Community structure detection via short random walks*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

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

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

Note

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

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

Author(s)

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

References

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

See Also

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

Associated functions: netclu_infomap netclu_louvain netclu_oslom

Examples

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

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

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

---

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

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

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

See Also

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

Associated functions: mat_to_net

Examples

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

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

---

nhclu_affprop *Non-hierarchical clustering: Affinity Propagation*

Description

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

Usage

```
nhclu_affprop(
  similarity,
  index = names(similarity)[3],
  seed = NULL,
  p = NA,
  q = NA,
  maxits = 1000,
  convits = 100,
```

```
  lam = 0.9,
  details = FALSE,
  nonoise = FALSE,
  K = NULL,
  prc = NULL,
  bimaxit = NULL,
  exact = NULL,
  algorithm_in_output = TRUE,
  verbose = TRUE
)
```

Arguments

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

`algorithm_in_output`
: A `boolean` indicating whether to include the original output of apcluster in the result. Defaults to `TRUE`.

`verbose`
: A `boolean` indicating whether to display progress messages. Set to `FALSE` to suppress these messages.

Details

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

Value

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

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

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

Author(s)

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

References

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

See Also

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

Associated functions: nhclu_clara nhclu_clarans nhclu_dbscan nhclu_kmeans nhclu_affprop

Examples

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

```
                                        paste0("Species", 13:20))))

comat_2 <- matrix(sample(0:1000,
                         size = 10*12,
                         replace = TRUE,
                         prob = 1/1:1001),
                  10, 12)
rownames(comat_2) <- paste0("Site", 11:20)
colnames(comat_2) <- paste0("Species", 9:20)
comat_2 <- cbind(matrix(0, 10, 8,
                        dimnames = list(paste0("Site", 11:20),
                                        paste0("Species", 1:8))),
                 comat_2)

comat <- rbind(comat_1, comat_2)

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

clust1 <- nhclu_affprop(sim)

clust2 <- nhclu_affprop(sim, q = 1)

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

`nhclu_clara` *Non-hierarchical clustering: CLARA*

Description

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

Usage

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

Arguments

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

Details

Based on fastkmedoids package (fastclara).

Value

A list of class bioregion.clusters with five components:

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

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

Author(s)

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

References

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

See Also

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

Associated functions: nhclu_clarans nhclu_dbscan nhclu_kmeans nhclu_pam nhclu_affprop

Examples

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

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

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

---

nhclu_clarans *Non-hierarchical clustering: CLARANS*

Description

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

Usage

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

Arguments

dissimilarity	The output object from dissimilarity() or similarity_to_dissimilarity(), or a dist object. If a data.frame is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
index	The name or number of the dissimilarity column to use. By default, the third column name of dissimilarity is used.
seed	A value for the random number generator (NULL for random initialization by default).
n_clust	An integer vector or a single integer specifying the desired number(s) of clusters.
numlocal	An integer defining the number of local searches to perform.
maxneighbor	A positive numeric value defining the maximum number of neighbors to consider for each local search.
algorithm_in_output	A boolean indicating whether the original output of fastclarans should be included in the output. Defaults to TRUE (see Value).

Details

Based on fastkmedoids package (fastclarans).

Value

A list of class bioregion.clusters with five components:

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

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

Author(s)

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

References

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

See Also

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

Associated functions: nhclu_clara nhclu_dbscan nhclu_kmeans nhclu_pam nhclu_affprop

Examples

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

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

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

---

nhclu_dbscan *Non-hierarchical clustering: DBSCAN*

Description

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

Usage

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

Arguments

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

minPts	A numeric vector or a single numeric value specifying the minPts argument of dbscan::dbscan(). minPts is the minimum number of points to form a dense region. By default, it is set to the natural logarithm of the number of sites in dissimilarity, rounded down to a whole number. See Details for guidance on choosing this parameter.
eps	A numeric vector or a single numeric value specifying the eps argument of dbscan::dbscan(). eps specifies how similar points should be to each other to be considered part of a cluster. See Details for guidance on choosing this parameter.
plot	A boolean indicating whether the k-nearest neighbor distance plot should be displayed.
algorithm_in_output	A boolean indicating whether the original output of dbscan::dbscan should be included in the output. Defaults to TRUE (see Value).
verbose	A boolean indicating whether to display progress messages. Set to FALSE to suppress these messages.
...	Additional arguments to be passed to dbscan() (see dbscan::dbscan).

Details

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

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

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

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

Value

A list of class bioregion.clusters with five components:

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

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

Author(s)

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

References

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

See Also

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

Associated functions: nhclu_clara nhclu_clarans nhclu_kmeans nhclu_pam nhclu_affprop

Examples

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

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

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

---

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

Description

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

Usage

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

```
  algorithm_in_output = TRUE
)
```

Arguments

dissimilarity	The output object from dissimilarity() or similarity_to_dissimilarity(), or a dist object. If a data.frame is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
index	The name or number of the dissimilarity column to use. By default, the third column name of dissimilarity is used.
seed	A value for the random number generator (NULL for random by default).
n_clust	An integer vector or a single integer value specifying the requested number(s) of clusters.
iter_max	An integer specifying the maximum number of iterations for the k-means method (see kmeans).
nstart	An integer specifying how many random sets of n_clust should be selected as starting points for the k-means analysis (see kmeans).
algorithm	A character specifying the algorithm to use for k-means (see kmeans). Available options are Hartigan-Wong, Lloyd, Forgy, and MacQueen.
algorithm_in_output	A boolean indicating whether the original output of kmeans should be included in the output. Defaults to TRUE (see Value).

Details

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

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

Value

A list of class bioregion.clusters with five components:

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

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

Author(s)

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

See Also

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

Associated functions: nhclu_clara nhclu_clarans nhclu_dbscan nhclu_pam nhclu_affprop

Examples

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

comnet <- mat_to_net(comat)

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

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

---

nhclu_pam *Non-hierarchical clustering: Partitioning Around Medoids*

Description

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

Usage

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

Arguments

dissimilarity	The output object from dissimilarity() or similarity_to_dissimilarity(), or a dist object. If a data.frame is used, the first two columns should represent pairs of sites (or any pair of nodes), and the subsequent column(s) should contain the dissimilarity indices.
index	The name or number of the dissimilarity column to use. By default, the third column name of dissimilarity is used.
seed	A value for the random number generator (NULL for random by default).
n_clust	An integer vector or a single integer value specifying the requested number(s) of clusters.
variant	A character string specifying the PAM variant to use. Defaults to faster. Available options are original, o_1, o_2, f_3, f_4, f_5, or faster. See pam for more details.
nstart	An integer specifying the number of random starts for the PAM algorithm. Defaults to 1 (for the faster variant).
cluster_only	A boolean specifying whether only the clustering results should be returned from the pam function. Setting this to TRUE makes the function more efficient.
algorithm_in_output	A boolean indicating whether the original output of pam should be included in the result. Defaults to TRUE (see Value).
...	Additional arguments to pass to pam() (see pam).

Details

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

Value

A list of class bioregion.clusters with five components:

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

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

Author(s)

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

References

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

See Also

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

Associated functions: nhclu_clara nhclu_clarans nhclu_dbscan nhclu_kmeans nhclu_affprop

Examples

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

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

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

---

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

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

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

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

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

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

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

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

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

Euclidean = 1 / (1 + d_ij)

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

Value

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

Author(s)

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

References

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

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

See Also

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

Associated functions: dissimilarity similarity_to_dissimilarity

Examples

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

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

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

---

similarity_to_dissimilarity

Convert similarity metrics to dissimilarity metrics

Description

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

Usage

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

Arguments

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

Value

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

Note

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

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

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

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

dissimilarity = 1 - similarity

Author(s)

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

See Also

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

Associated functions: dissimilarity similarity_to_dissimilarity

Examples

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

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

dissimilarity <- similarity_to_dissimilarity(simil)
dissimilarity
```

site_species_metrics *Calculate metrics for sites and species relative to bioregions and chorotypes*

Description

This function computes metrics that quantify how species and sites relate to clusters (bioregions or chorotypes). Depending on the type of clustering, metrics can measure how species are distributed across bioregions (site clusters), how sites relate to chorotypes (species clusters), or both.

Usage

```
site_species_metrics(
  bioregionalization,
  bioregion_metrics = c("specificity", "n_specificity", "fidelity", "ind_val",
    "n_ind_val", "rho"),
  bioregionalization_metrics = "p",
  data_type = "auto",
  cluster_on = "site",
  comat,
  similarity = NULL,
  include_cluster = FALSE,
  index = names(similarity)[3],
  verbose = TRUE
)
```

Arguments

`bioregionalization`
: A `bioregion.clusters` object.

`bioregion_metrics`
: A `character` vector or a single `character` string specifying the metrics to compute for each cluster. Available metrics depend on the type of clustering (see arg `cluster_on`):
  - **When sites are clustered into bioregions** (default case): species-level metrics include `"specificity"`, `"n_specificity"`, `"fidelity"`, `"ind_val"`, `"n_ind_val"`, `"rho"`, and `"core_terms"`. Site-level metrics include `"richness"`, `"rich_endemics"`, `"prop_endemics"`, `"mean_sim"`, and `"sd_sim"`.
  - **When species are clustered into chorotypes** (e.g., bipartite network clustering): site-level metrics include `"specificity"`, `"n_specificity"`, `"fidelity"`, `"ind_val"`, `"n_ind_val"`, `"rho"`, and `"core_terms"`.

  Use `"all"` to compute all available metrics. See Details for metric descriptions.

`bioregionalization_metrics`
: A `character` vector or a single `character` string specifying summary metrics computed across all clusters. These metrics assess how an entity (species or site) is distributed across the entire bioregionalization, rather than relative to each individual cluster:

- "p": Participation coefficient measuring how evenly a species or site is distributed across clusters (0 = restricted to one cluster, 1 = evenly spread).
- "silhouette": How well a site fits its assigned bioregion compared to the nearest alternative bioregion (requires similarity data).

Use "all" to compute all available metrics.

data_type A character string specifying whether metrics should be computed based on presence/absence ("occurrence") or abundance values ("abundance"). This affects how specificity, fidelity, ind_val, rho and core_terms are calculated:

- "auto" (default): Automatically detected from input data (bioregionalization and/or comat).
- "occurrence": Metrics based on presence/absence only.
- "abundance": Metrics weighted by abundance values.
- "both": Compute both versions of the metrics.

cluster_on A character string specifying what was clustered in the bioregionalization, which determines what types of metrics can be computed:

- "site" (default): Sites were clustered into bioregions. Metrics describe how each **species** is distributed across bioregions.
- "species": Species were clustered into chorotypes. Metrics describe how each **site** relates to chorotypes. Only available when species have been assigned to clusters (e.g., bipartite network clustering).
- "both": Compute metrics for both perspectives. Only available when both sites and species have cluster assignments.

comat A site-species matrix with sites as rows and species as columns. Values can be occurrence (1/0) or abundance. Required for most metrics.

similarity A site-by-site similarity object from similarity() or dissimilarity_to_similarity(). Required only for similarity-based metrics ("mean_sim", "sd_sim", "silhouette").

include_cluster A boolean indicating whether to add an Assigned column in the output, marking TRUE for rows where the site belongs to the bioregion being evaluated. Useful for quickly identifying a site's own bioregion. Default is FALSE.

index The name or number of the column to use as similarity. By default, the third column name of similarity is used.

verbose A boolean indicating whether to display progress messages. Set to FALSE to suppress these messages.

Details

This function computes metrics that characterize the relationship between species, sites, and clusters. The available metrics depend on whether you clustered sites (into bioregions) or species (into chorotypes).

— 1. Understanding the two perspectives —:

- **Bioregions** are clusters of sites with similar species composition.
- **Chorotypes** are clusters of species with similar distributions.

In general, the package is designed to cluster sites into bioregions. However, it is possible to group species into clusters. We call these species clusters 'chorotypes', following conceptual definitions in the biogeographical literature, to avoid any confusion in the calculation of metrics.

In some cases, such as bipartite network clustering, both species and sites receive the same clusters. We maintain the name distinction in the calculation of metrics - but remember that in this case BIOREGION IDs = CHOROTYPE IDs. The `cluster_on` argument determines which perspective to use.

— 2. Metrics when sites are clustered (`cluster_on = "site"` **or** `cluster_on = "both"`) **—:**

Species-per-bioregion metrics quantify how each species is distributed across bioregions.

These metrics are derived from three core terms (see the online vignette for a visual diagram):

- **n_sb**: Number of sites in bioregion **b** where species **s** is present
- **n_s**: Total number of sites in which species **s** is present.
- **n_b**: Total number of sites in bioregion **b**.

Abundance version of these core terms can also be calculated when `data_type = "abundance"` (or `data_type = "auto"` and `bioregionalization was based on abundance`):

- **w_sb**: Sum of abundances of species **s** in sites of bioregion **b**.
- **w_s**: Total abundance of species **s**.
- **w_b**: Total abundance of all species present in sites of bioregion **b**.

The species-per-bioregion metrics are (click on metric names to access formulas):

- **specificity**: Fraction of a species' occurrences found in a given bioregion (De Cáceres & Legendre 2009). A value of 1 means the species occurs only in that bioregion.
- **n_specificity**: Normalized specificity that accounts for differences in bioregion size (De Cáceres & Legendre 2009).
- **fidelity**: Fraction of sites in a bioregion where the species occurs (De Cáceres & Legendre 2009). A value of 1 means the species is present in all sites of that bioregion.
- **ind_val**: Indicator value (ind_val = specificity × fidelity) as defined in (De Cáceres & Legendre 2009). High values identify species that are both restricted to and frequent within a bioregion.
- **n_ind_val**: Normalized ind_val accounting for bioregion size (De Cáceres & Legendre 2009).
- **rho**: Standardized contribution index comparing observed vs. expected co-occurrence under random association (Lenormand 2019).
- **core_terms**: Raw counts (n, n_b, n_s, n_sb) for custom calculations.

These metrics can be found in the output slot `species_bioregions`.

Site-per-bioregion metrics characterize sites relative to bioregions:

- **richness**: Number of species in the site.
- **rich_endemics**: Number of species in the site that are endemic to one bioregion.
- **prop_endemics**: Proportion of endemic species in the site.
- **mean_sim**: Mean similarity of a site to all sites in each bioregion.
- **sd_sim**: Standard deviation of similarity values.

These metrics can be found in the output slot `site_bioregions`.

Summary metrics across the whole bioregionalization:

These metrics summarize how an entity (species or site) is distributed across all clusters, rather than in relation to each individual cluster.

Species-level summary metric:

- **p** (Participation): Evenness of species distribution across bioregions (Denelle et al. 2020). Found in output slot `species_bioregionalization`.

Site-level summary metric:

- **silhouette**: How well a site fits its assigned bioregion vs. the nearest alternative (Rousseeuw 1987). Found in output slot `site_bioregionalization`.

— 3. Metrics when species are clustered (`cluster_on = "species"` or `cluster_on = "both"`) —:

Site-per-chorotype metrics quantify how each site relates to species clusters (chorotypes).
The same metrics as above (specificity, fidelity, ind_val, etc.) can be computed, but their interpretation is inverted. These metrics are based on the following core terms:

- **n_gc**: Number of species belonging to chorotype **c** that are present in site **g**.
- **n_g**: Total number of species present in site **g**.
- **n_c**: Total number of species belonging to chorotype **c**.

Abundance version of these core terms can also be calculated when `data_type = "abundance"` (or `data_type = "auto"` and `bioregionalization was based on abundance`).
Their interpretation changes, for example:

- **specificity**: Fraction of a site's species belonging to a chorotype.
- **fidelity**: Fraction of a chorotype's species present in the site.
- **ind_val**: Indicator value for site-chorotype associations.
- **p**: Evenness of sites across chorotypes

Value

A `list` containing one or more `data.frame` elements, depending on the selected metrics and clustering type:

When sites are clustered (`cluster_on = "site"`):

- **species_bioregions**: Metrics for each species x bioregion combination (e.g., specificity, ind_val). One row per species x bioregion pair.
- **species_bioregionalization**: Summary metrics for each species across all bioregions (e.g., Participation coefficient). One row per species.
- **site_bioregions**: Metrics for each site x bioregion combination (e.g., mean_sim, richness). One row per site x bioregion pair.
- **site_bioregionalization**: Summary metrics for each site (e.g., silhouette). One row per site.

When species are clustered (`cluster_on = "species"`):

- **site_chorotypes**: Metrics for each site x chorotype combination (e.g., specificity, ind_val). One row per site x chorotype pair.
- **site_chorological**: Summary metrics for each site across all chorotypes (e.g., Participation coefficient). One row per site.

Note that if `bioregionalization` contains multiple partitions (i.e., if `dim(bioregionalization$clusters) > 2`), a nested list will be returned, with one sublist per partition.

Note

If `data_type = "auto"`, the choice between occurrence- or abundance- based metrics will be determined automatically from the input data, and a message will explain the choice made.

Strict matching between entity IDs (site and species IDs) in `bioregionalization` and in `comat` / `similarity` is required.

Author(s)

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

References

De Cáceres M & Legendre P (2009) Associations between species and groups of sites: indices and statistical inference. *Ecology* 90, 3566–3574.

Denelle P, Violle C & Munoz F (2020) Generalist plants are more competitive and more functionally similar to each other than specialist plants: insights from network analyses. *Journal of Biogeography* 47, 1922—1933.

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

Rousseeuw PJ (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* 20, 53–65.

See Also

For more details illustrated with a practical example, see the vignette: https://biorgeo.github.io/bioregion/articles/a5_2_summary_metrics.html.

Associated functions: bioregion_metrics bioregionalization_metrics

Examples

```
data(fishmat)

fishsim <- similarity(fishmat, metric = "Jaccard")

bioregionalization <- hclu_hierarclust(similarity_to_dissimilarity(fishsim),
                                       index = "Jaccard",
                                       method = "average",
                                       randomize = TRUE,
                                       optimal_tree_method = "best",
                                       n_clust = c(1,2,3),
                                       verbose = FALSE)

ind <- site_species_metrics(bioregionalization = bioregionalization,
                            bioregion_metrics = "all",
                            bioregionalization_metrics = "all",
                            data_type = "auto",
```

```
cluster_on = "site",
comat = fishmat,
similarity = fishsim,
include_cluster = TRUE,
index = 3,
verbose = TRUE)
```

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

Description

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

Usage

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

Arguments

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

Value

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

Note

Some `bioregion.clusters` objects may contain both types of nodes (sites and species). This information is available in the `$inputs$node_type` slot.

This function allows you to extract a specific type of node (either sites or species) from any `bioregion.clusters` object that includes both.

Author(s)

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

Examples

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

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

clu_sites <- site_species_subset(clu, node_type = "site")
```

---

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

Description

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

Usage

```
vegedf
```

Format

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

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

Species Unique species identifier

Abundance Species abundance

Source

doi:10.1002/ece3.4718

---

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

Description

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

Usage

```
vegemat
```

Format

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

Source

doi:10.1002/ece3.4718

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

Description

A dataset containing the geometry of the 715 sites.

Usage

```
vegesf
```

Format

A `sf` `data.frame` with 715 rows and 2 columns:

Site Unique site identifier

geometry Geometry of the site

Source

doi:10.1002/ece3.4718

Index

∗ **datasets**
fishdf, 29
fishmat, 29
fishsf, 30
vegedf, 99
vegemat, 99
vegesf, 100

apcluster, *78*
apclusterK, *77*
as_bioregion_pairwise, 3, *5, 6*

bcdist, *4*
beta.div, *4*
beta.div.comp, *4*
beta.pair, *4*
beta.pair.abund, *4*
betapart.core, *4*
betapart.core.abund, *4*
betapart_to_bioregion, 5
bind_pairwise, *4*, 6
bioregion_colors, 9, *50*
bioregion_colors(), *23*
bioregion_metrics, 12, *97*
bioregionalization_metrics, 7, *13*, *16*, *31*, *33*, *97*
bioregionalization_metrics(), *17*, *26*, *31*, *35*

cluster_fast_greedy, *55*
cluster_label_prop, *60*
cluster_leading_eigen, *62, 63*
cluster_leiden, *65*
cluster_louvain, *68, 69*
cluster_walktrap, *73, 74*
compare_bioregionalizations, *9*, 14
computeModules, *53*
consensus, *35*
cut_tree, 17, *32*, *38*, *46*
cut_tree(), *46*
cutree, *46*

dbscan, *39, 40*
dbscan::dbscan, *84*
dbscan::dbscan(), *84*
designdist, *4*
diana, *31*
dissimilarity, *4*, *6*, 19, *91, 92*
dissimilarity(), *7*, *21, 22*, *31*, *33*, *39*, *80*, *82, 83*, *86*, *88*
dissimilarity_to_similarity, *21*, 21, *22*
dissimilarity_to_similarity(), *7*, *54*, *57*, *60*, *62*, *64*, *67*, *70*, *73*, *77*, *92*, *94*
distance, *4*
dynamicTreeCut::cutreeDynamic(), *18*

exportGDF, 23
extractXi, *40*

fastclara, *80*
fastclarans, *82*
find_optimal_n, *9*, 26
fishdf, 29
fishmat, 29
fishsf, 30

hclu_diana, 30
hclu_hierarclust, *19*, *28*, 32, *46*
hclu_hierarclust(), *10*, *18*, *41*
hclu_optics, 39
hclust, *33*, *35*, *42, 43*

ihct, *38*, 41
ihct(), *36*
install_binaries, 47, *57, 58*, *67, 68*, *71*

kmeans, *86*

map_bioregions, *11*, 48
map_bioregions(), *11*
mat_to_net, 50, *76*

net_to_mat, *51*, 75
netclu_beckett, 52
netclu_greedy, 54, *59*, *69*, *72*
netclu_greedy(), *23*
netclu_infomap, *48*, *53*, *56*, 56, *61*, *63*, *66*, *69*, *72*, *74*
netclu_infomap(), *10*
netclu_labelprop, 59
netclu_leadingeigen, 61
netclu_leiden, 64
netclu_louvain, *48*, *53*, *56*, *59*, *61*, *63*, *66*, 66, *72*, *74*
netclu_oslom, *48*, *53*, *56*, *59*, *61*, *63*, *66*, *69*, 70, *74*
netclu_walktrap, 73
nhclu_affprop, 76, *78*, *81*, *83*, *85*, *87*, *89*
nhclu_clara, *78*, 79, *83*, *85*, *87*, *89*
nhclu_clarans, *78*, *81*, 81, *85*, *87*, *89*
nhclu_dbscan, *41*, *78*, *81*, *83*, 83, *87*, *89*
nhclu_kmeans, *78*, *81*, *83*, *85*, 85, *89*
nhclu_pam, *81*, *83*, *85–87*, 87
nhclu_pam(), *10*
nnls.tree, *35*

optics, *40*

pam, *88*
pcoa, *86*

set.seed(), *44*
similarity, *4*, *6*, *21, 22*, 89
similarity(), *54*, *57*, *60*, *62*, *64*, *67*, *70*, *73*, *77*, *92*, *94*
similarity_to_dissimilarity, *91*, 91, *92*
similarity_to_dissimilarity(), *21*, *31*, *33*, *39*, *80*, *82, 83*, *86*, *88*
site_species_metrics, *13*, 93
site_species_subset, 98

vegan::anosim(), *8*
vegdist, *4*
vegedf, 99
vegemat, 99
vegesf, 100