Package 'hespdiv'

May 21, 2026

Type Package

Title Hierarchical Spatial Data Subdivision into Topologically Contiguous Units

Version 1.2.10

Maintainer Liudas Daumantas <liudas.daumantas@chgf.vu.lt>

Description
Implementation of the HespDiv framework for hierarchical spatial subdivision of geographical occurrence data. The main function hespdiv() performs iterative spatially constrained subdivision of a study area to identify topologically contiguous clusters in geographic space using user-defined or preset subdivision methods. Additional functions provide tools for analysing subdivision results, visualizing hierarchical spatial structures, and evaluating robustness through sensitivity analyses and statistical testing. Some examples use the optional HDData data package, which is available from GitHub at Liudas-Dau/hespdiv_data. The methodology is described in Daumantas and Spiridonov (2024) <doi:10.1111/pala.12702>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

OS_type windows

LazyData true

URL https://doi.org/10.1111/pala.12702,
https://github.com/Liudas-Dau/hespdiv,
https://github.com/Liudas-Dau/hespdiv_data

Imports pracma, DescTools, ggplot2, viridis, ggrepel, grDevices, RColorBrewer, rgl, scales, graphics, igraph, gridExtra, grid, gridGraphics, stats, magick, future.apply, future, rlang

Suggests HDData

Depends R (>= 4.0)

NeedsCompilation yes

Author Liudas Daumantas [aut, cre], Andrej Spiridonov [aut],

Joseph O'Rourke [ctb, cph] (Author and copyright holder of
point-in-polygon code in src/pip.c),
Min Xu [ctb] (Contributor to point-in-polygon code in src/pip.c)

Repository CRAN

Date/Publication 2026-05-21 13:10:02 UTC

Contents

`blok3d` *Draw hespdiv polygons in 3D space*

Description

This function visualizes HespDiv polygons in 3D space. The height axis corresponds to a chosen column from the "poly.stats" data frame.

Usage

```
blok3d(
  obj,
  height = "mean",
  color.seed = 1,
  lines = TRUE,
  pnts.col = NULL,
  obs = TRUE
)
```

Arguments

obj	An object of the hespdiv class
height	A character vector with default value "mean". Determines which information from the poly.stats data frame is encoded as the height of the polygons. Options are "mean", "sd", "best", "z.score", "str.best", "str.z.score", and "rank". Multiple values are allowed.
color.seed	An integer that controls the colors of the polygons. Change it to a different number if you want to get a different set of colors.
lines	A Boolean value. Do you want split-lines to be displayed over the top of the polygons?
pnts.col	A character or numeric vector. Color codes to be used for displaying observations.
obs	A Boolean value. Do you want observations to be displayed over the top of the polygons?

Details

The function opens an **rgl** device for each selected column of the poly.stats data frame.

Visualizing values from poly.stats as polygon height can provide insight into spatial heterogeneity in the analyzed data and its hierarchical spatial structure.

The height = "rank" option provides a more intuitive way to understand the position of each polygon in the spatial hierarchy than poly_scheme().

Because higher-rank polygons are displayed above lower-rank polygons, they may obscure the view. For this reason, polypop(obj, height) can be used with the same arguments to interactively select unwanted polygons and remove them from the actives plot.

Value

No return value. Called for the side effect of producing a 3D plot using the **rgl** graphics engine.

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: create_gif(), dendro(), plot.nullhespdiv(), plot_cs_hsa(), plot_hespdiv(), plot_hsa(), plot_hsa_q(), poly_scheme(), polypop()

---

change_base *Change the basal subdivision in hsa*

Description

This function allows you to select a new basal subdivision from the results of hespdiv sensitivity analysis by specifying its ID. It provides a convenient way to switch between different basal subdivisions. You can identify a more stable subdivision alternative by plotting the hespdiv sensitivity analysis results using the plot_hsa function. By selecting a new basal subdivision, you can observe how it affects the results of polygon object cross-comparison and the stability of hespdiv clusters and polygons.

Usage

```
change_base(obj, id)
```

Arguments

obj	A hsa class object.
id	An index of an alternative subdivision to be used as a new basal subdivision.

Value

The hsa class object with a new basal subdivision.

Author(s)

Liudas Daumantas

See Also

Other functions for hespdiv sensitivity analysis: hsa(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), plot_cs_hsa(), plot_hsa(), plot_hsa_q()

`create_gif` *Create a gif of blok3D*

Description

Functions creates a gif of rotating polygons displayed in a currently actively rgl device produced with blok3D

Usage

```
create_gif(output_file, frames = 90, angle_per_frame = 5, fps = 10)
```

Arguments

`output_file`	name and path of output file
`frames`	integer. Number of frames.
`angle_per_frame`	numeric. Angle to rotate per frame in degrees.
`fps`	integer. Frames per second.

Value

No return value. Called for the side effect of saving a GIF of a rotating 3D plot.

Note

You can adjust the size of rgl device window to control the size of gif.

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: `blok3d()`, `dendro()`, `plot.nullhespdiv()`, `plot_cs_hsa()`, `plot_hespdiv()`, `plot_hsa()`, `plot_hsa_q()`, `poly_scheme()`, `polypop()`

`cross_comp` *Calculate a polygon-object cross-comparison matrix*

Description

Computes a cross-comparison matrix for polygon objects from a `hespdiv` object. The matrix quantifies similarity or dissimilarity among polygon objects and can be used for further analyses, such as clustering, either directly or after transformation.

Usage

```
cross_comp(obj)
```

Arguments

`obj`	A `hespdiv` object.

Details

The `cross_comp()` function uses the `compare.f` function from `obj$call.info$Call_ARGS` to perform pairwise comparisons of `hespdiv` polygon objects stored in `obj$poly.obj`. The result is a cross-comparison matrix.

Value

A numeric matrix containing pairwise comparison values among the `hespdiv` polygon objects stored in `obj$poly.obj`.

Note

Polygon cross-comparison is currently not available for the `"pielou"` method. It is also not supported for custom methods whose `compare.f` function relies on variables from environments other than the function's own arguments.

Author(s)

Liudas Daumantas

See Also

Other functions for hespdiv results post-processing: `hsa()`, `hsa_detailed()`, `hsa_quant()`, `hsa_sample_constrained()`, `nulltest()`, `taxon_effect()`

dendro *Plot hespdiv dendrogram*

Description

This function displays a dendrogram of polygons produced by hespdiv split-lines. Branch length is proportional to difference. If performance of split-lines is a similarity measure, it is internally converted to difference.

Usage

```
dendro(
  obj,
  poly.scheme = NULL,
  color = 1,
  performance.col = "blue",
  labels.col = 1,
  offset.factor = 1,
  arrange = TRUE,
  grob = TRUE,
  label.size = 0.5
)
```

Arguments

obj	A hespdiv object.
poly.scheme	ggplot2 object produced with poly_scheme function. Provide if you want identical colors for polygons in both plots.
color	color vector used for dendrogram nodes and branches.
performance.col	color vector used for text, displaying difference values between polygons
labels.col	color vector used for text, displaying polygon IDs.
offset.factor	numeric value used to scale the offset distance of displayed polygon IDs and performance values from a dendrogram node. Adjust experimentally, if you don't like the current distance.
arrange	logical. Plot hespdiv dendrogram above polygon scheme?
grob	logical. Convert plot to grob? Must be true, if you want to arrange polygon scheme and the dendrogram in a single plot.
label.size	size of labels.

Value

A grob or TableGrob object if grob = TRUE, otherwise NULL.

Note

If you want to transform similarity to difference externally, before applying dendro, change maximize to TRUE in the call info of obj.

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: `blok3d()`, `create_gif()`, `plot.nullhespdiv()`, `plot_cs_hsa()`, `plot_hespdiv()`, `plot_hsa()`, `plot_hsa_q()`, `poly_scheme()`, `polypop()`

Examples

```
scheme <- poly_scheme(example_hespdiv)
# notice the colors in used in scheme:
scheme
# Dendrogram visualization of polygons, using colors from scheme
dendro(example_hespdiv, poly.scheme = scheme, arrange = FALSE, grob = FALSE)
```

---

depend_splits *Identify dependent split-lines and polygons*

Description

For a provided split-line id, function returns a list of all offspring polygons and split-lines, if there are any.

Usage

```
depend_splits(obj, id)
```

Arguments

`obj`	hespdiv object
`id`	id of a split-line

Value

a list:

1. Dependent split-line IDs, if any
2. Dependent polygon IDs

example_hespdiv *Example hespdiv object*

Description

A small hespdiv object created from simulated occurrence data with a known boundary at x = 0.45. The object is intended for examples, tests, and demonstrations of plotting and post-processing functions.

Format

A hespdiv object.

Details

The object was generated using simulated coordinates and taxon names. Occurrences on the left and right sides of the artificial boundary share several common taxa but also include side-specific taxa. The exact code used to simulate the dataset and run hespdiv is also demonstrated in the examples of hespdiv function.

Examples

```
data(example_hespdiv)
plot_hespdiv(example_hespdiv)
```

get_data *Get data points that lie inside a polygon*

Description

This function extracts points from a provided dataset that lie inside a given polygon.

Usage

```
get_data(polygon, xy.dat)
```

Arguments

polygon	A data frame of 2 columns ("x","y") that contain coordinates of polygon vertices.
xy.dat	A data frame, containing "x" and "y" columns. Other columns may also be present.

Value

A filtered data frame.

Note

This function excludes points that are strictly outside the given polygon. Points lying on the edges or vertices of the polygon (if the polygon is not closed) will be included in the filtered data frame.

Author(s)

Liudas Daumantas

Examples

```
#Creating data.frame of a polygon
poly<- data.frame(c(3.38,3.30,1.70,0.78,-0.06,-2.30,-2.94,-3.97,-1.61,-0.39,0.68,1.28,1.60,3.38),
c(-0.12,-0.31,-2.73,-3.22,-3.29,-2.19,-1.62,0.94,3.10,3.00,2.91,2.49,2.20,-0.12))

#Creating a data set of points
xy.dat<-data.frame(x=runif(250,-4,4),y=runif(250,-4,4))
plot(poly,type='l',xlab="X",ylab="Y")
points(xy.dat)

#Extracting points that lie inside a polygon
points(get_data(poly,xy.dat),pch=19,col=2)
```

`group_effect` *Investigate Group Effects on Split-Line Performance*

Description

For every level of `group`, the function:

1. subsets `data` and `xy.dat` to the focal group's observations;
2. recomputes split-line performance using `compare.f` and `generalize.f`;
3. runs two contribution assessments:
   - *group-removal*: removes the group's observations and recomputes performance;
   - *group-permutation (locality-block shuffle)*: shuffles intact within-locality assemblages of the focal group among the set of localities where the group occurs within the split's parent polygons (non-group observations fixed).
4. optionally re-plots the object for that group via `plot_hespdiv` (with a group-specific subtitle).

Usage

```
group_effect(obj, group, perm.n = 999, maxdif = NULL, plot = TRUE, ...)
```

Arguments

obj	A hespdiv object.
group	A factor (or coercible to factor) giving the group label for each observation (row) in obj$call.info$Call_ARGS$xy.dat.
perm.n	Integer (> 0). Number of permutations in the locality-block permutation test. Default 999.
maxdif	Numeric. The performance value representing a maximal between-polygons difference for the chosen metric. If NULL and the metric is one of c("pielou","morisita","sorensen","h maxdif is set automatically. Otherwise, it must be provided.
plot	Logical. If TRUE (default), stores a plot_hespdiv output per group with performance = TRUE and a group-specific subtitle.
...	Additional arguments passed to plot_hespdiv. Note that obj, performance, and subtitle are set internally.

Details

Re-evaluates split-line performance within each level of a grouping factor and tests how much each group influences the detected split-lines in a hespdiv object.

Within-group recomputation (agreement). For each split, the group's observations are partitioned by the two child polygons and performance is recomputed as compare.f(generalize.f(pol1), generalize.f(pol2)). If the group's points fall on only one side, within$est is set to maxdif; if absent from both sides, it is NA.

Elimination test. All observations of the focal group are removed from both polygons and performance is recomputed on the remaining data. If the group's points fall on only one side, elim$est is computed normally; if absent from both sides, it is NA (if desired, you could identify these cases afterwards from n_per_pol, changing to baseline performance and zero delta).

Permutation test (locality-block shuffle). Localities are defined by identical coordinate pairs among the focal group's occurrences. For each permutation, whole localities (blocks) are reassigned via a one-to-one mapping within the split's parent polygons. For speed, polygon membership of unique locality coordinates is precomputed once per split and then locality memberships are shuffled in each permutation. If the group's points fall on only one side, perm$est is set to NA; if absent from both sides, it is also NA.

Baseline and deltas. The baseline vector is baseline <- obj$split.stats$performance. Let maximize <- obj$call.info$Call_ARGS$maximize. Deltas are defined so that *positive values always indicate a positive contributor* (removal/permutation worsens performance):

- if maximize == FALSE (lower is better; e.g. similarity): delta = est - baseline;
- if maximize == TRUE (higher is better; e.g. distance): delta = baseline - est.

Value

A list of class "group_effect" with elements:

- within = list(est, delta) where est and delta are [n_splits x n_groups] matrices;
- elim = list(est, delta) as above (group removed);

- `perm` = `list(est, delta)` where each is a nested list `[[split]][[group]]` of numeric vectors (length `perm.n`). If permutations are uninformative (e.g., one-sided/absent), the corresponding entries are `NA`;
- `baseline`: the original performance vector;
- `n_per_pol`: `data.frame` with columns `split.id`, `group`, `pol.id`, `n`;
- `plots`: list of `plot_hespdiv` outputs by group (or `NULL` if `plot` = `FALSE`).

Notes

- Localities are defined by exact duplicate coordinate pairs among the group's observations (harmonise coordinates upstream if needed).
- The locality-block permutation preserves the *number of group-present localities per polygon*; only identities of the locality blocks are shuffled. The number of group occurrences per side can vary if block sizes differ.

See Also

`plot_hespdiv`

Examples

```
data <- example_hespdiv$call.info$Call_ARGS$data
# See the hespdiv() example; lowercase letters represent widespread taxa:
endemic_l <- !data %in% letters[1:5]
group <- factor(ifelse(endemic_l, "endemic", "widespread"))

gr_ef <- group_effect(obj = example_hespdiv, group = group, plot = FALSE)

barplot(
  gr_ef$within$est,
  main = "Split-line similarity within each group",
  ylab = "Performance"
)

barplot(
  gr_ef$within$delta,
  main = "Group contribution to similarity within each group",
  ylab = "Contribution"
)
abline(h = 0)

barplot(
  gr_ef$elim$est,
  main = "Split-line similarity with group eliminated",
  ylab = "Performance"
)

barplot(
  gr_ef$elim$delta,
  main = "Group contribution to similarity with group eliminated",
```

```
  ylab = "Contribution"
)
abline(h = 0)

boxplot(
  gr_ef$perm$est$`1`,
  main = "Split-line similarity with group spatially permuted",
  ylab = "Performance"
)

boxplot(
  gr_ef$perm$delta$`1`,
  main = "Group contribution to similarity with group spatially permuted",
  ylab = "Contribution"
)
abline(h = 0)
```

`hespdiv` *Hierarchically subdivide spatial data*

Description

This function is an implementation of spatial data analysis method HespDiv. It performs hierarchical spatial data subdivision by recursively dividing the data using random split-lines, evaluating their comparison values (how well they separate data), and using the best to perform subdivisions.

Usage

```
hespdiv(
  data,
  xy.dat = NULL,
  n.split.pts = 15,
  same.n.split = TRUE,
  method = "horn.morisita",
  generalize.f = NULL,
  compare.f = NULL,
  maximize = NULL,
  N.crit = 1,
  N.rel.crit = 0.2,
  N.loc.crit = 1,
  N.loc.rel.crit = 0.2,
  S.crit = 0.05,
  S.rel.crit = 0.2,
  Q.crit = NULL,
  c.splits = TRUE,
  c.Q.crit = NULL,
  c.crit.improv = 0,
  c.X.knots = 5,
```

```
  c.Y.knots = 10,
  c.max.iter.no = +Inf,
  c.fast.optim = FALSE,
  c.corr.term = 0.05,
  study.pol = NULL,
  use.chull = TRUE,
  tracing = NULL,
  pnts.col = 1,
  display = FALSE,
  pacific.region = FALSE,
  .do_recurse = TRUE
)
```

Arguments

data	An R object that contains the data to be analyzed. The required data structure depends on the selected method (e.g., character vector of taxa names).
xy.dat	A data.frame containing the coordinates for observations in the data. This parameter can be ignored if data is a data frame or matrix with columns x or y.
n.split.pts	integer. The number of split-points - 1. These points are used in creating straight split-lines (see details). The total number of straight split-lines generated can be obtained by sum(1:n.split.pts). Increasing the value of n.split.pts leads to an increase in both the computation time and the fit to the data.
same.n.split	logical. Should the number of split-points (n.split.pts) remain the same within each lower-ranked polygon? Choosing TRUE would result in a higher density of straight split-lines within lower-ranked polygons, whereas FALSE would preserve the same split-line density. Thus, essentially you are choosing between cross-scale and fixed-scale analysis.
method	Character. Name or abbreviation of a preset method: "sorensen", "pielou", "morisita", or "horn.morisita". Internally determines values for compare.f, generalize.f, and maximize.
generalize.f	Function. Optional function used in custom methods to prepare input for the compare.f function; see Details. It must have a single argument, for example x, to which a spatially filtered subset of data can be assigned; see Note for exceptions.
compare.f	function. Only required in custom methods. Employed to quantify the comparison value of a split-line. For this purpose, the compare.f function requires two arguments where the outputs of the generalize.f function can be assigned (see notes for exceptions).
maximize	logical. Only required in custom methods. Determines whether the split-line comparison value should be maximized or minimized during the optimization process.
N.crit	number. Minimum required number of observations that should be present in each polygon obtained by a split-line in order for it to meet this criterion.
N.rel.crit	number from 0 to 0.5. Each polygon obtained with a split-line must have at least such proportion of observations to pass this criterion. Equation of the pro-

portion: (Number of observations in 1st/2nd resulting polygon) / (Number of observations in the polygon being subdivided)

`N.loc.crit` number. Minimum required number of different locations that should be present in each polygon obtained by a split-line in order for it to meet this criterion.

`N.loc.rel.crit` number from 0 to 0.5. Each polygon obtained with a split-line must have at least such proportion of different locations to pass this criterion. Equation of the proportion: (Number of different locations in 1st/2nd resulting polygon) / (Number of different locations in the polygon being subdivided)

`S.crit` number from 0 to 1. Each polygon obtained with a split-line must have at least such area proportion to pass this criterion. Equation of the proportion: (Area of 1st/2nd resulting polygon) / (Area of the first polygon). The first polygon is the provided study area or convex hull of observation locations.

`S.rel.crit` number from 0 to 0.5. Each polygon obtained with a split-line must have at least such area proportion to pass this criterion. Equation of the proportion: (Area of 1st/2nd resulting polygon) / (Area of the polygon being subdivided).

`Q.crit` number. The threshold for a split-line comparison value to be considered acceptable for a subdivision. When `maximize = TRUE`, higher values of the comparison value indicate a better subdivision. Conversely, when `maximize = FALSE`, lower values of the comparison value indicate a better subdivision.

`c.splits` logical. When set to TRUE, the algorithm will explore nonlinear split-lines in addition to straight split-lines in order to find the optimal subdivision.

`c.Q.crit` number. The threshold for a split-line comparison value to be considered acceptable for generating nonlinear split-lines. It is recommended to use the default value, which does not impose a performance requirement, unless you have a clear understanding of the potential improvements that nonlinear split-lines can achieve over straight split-lines. If `maximize = TRUE`, a suggested value for `c.Q.crit` is `Q.crit` minus the maximum potential improvement.

`c.crit.improv` number. The threshold for the improvement in a split-line comparison value required for a nonlinear split-line to be selected instead of a straight split-line for subdivision. The default value of 0 means that even if a nonlinear split-line performs equally to a straight split-line, the straight split-line will still be chosen.

`c.X.knots` integer. Specifies the number of columns in a network of spline knots used to generate nonlinear split-lines. These knots are evenly distributed along the straight split-line. Adjusting the value of `c.X.knots` controls the degree of wiggliness (number of turns) in the resulting nonlinear split-lines.

`c.Y.knots` integer. specifies the number of rows in a network of spline knots used to generate nonlinear split-lines. These knots are distributed regularly along lines orthogonal to the straight split-line. Adjusting the value of `c.Y.knots` controls the number of amplitudes tested in each wiggle of a spline, influencing the shape of nonlinear split-lines.

`c.max.iter.no` integer. The maximum number of iterations allowed through the network of spline knots when searching for the optimal shape of a nonlinear split-line. Setting a higher value, such as `+Inf`, increases the chances of converging to the best possible curve. It is recommended to use higher values when `c.fast.optim = FALSE`, as in this case, only a single spline knot can be selected per iteration.

`c.fast.optim` logical. Determines when spline knots are selected. If `TRUE`, the algorithm selects the first knot that generates a curve with a better comparison value, ensuring faster convergence. If set to `FALSE`, the algorithm completes a full iteration through the network of spline knots, which may result in slower convergence but potentially better overall performance.

`c.corr.term` number from 0.01 to 0.2. A correction term for nonlinear split-lines that intersect the boundary of the polygon. Smaller values (default is 0.05) are recommended, as they determine the extent to which the outlying interval of the generated spline, which crosses the polygon boundary, should be shifted away from the boundary and inside the polygon in a direction orthogonal to the straight split-line. This shift is specified as a proportion of the polygon height where the spline intersects the polygon boundary.

`study.pol` A data frame with two columns, `x` and `y`. Rows should correspond to vertices of a polygon that defines the study area encompassing all observation locations in `xy.dat`. If not provided (default is `NULL`), the convex hull of `xy.dat` is used as the study area polygon.

`use.chull` logical. If `study.pol` is provided, you can use `use.chull = TRUE` (which is default) to use it only for visualization.

`tracing` Optional character vector of length two controlling diagnostic tracing output. The first element specifies the tracing level and must be one of `"best"`, `"main"`, or `"all"`. The second element specifies the traced object and must be one of `"curves"`, `"straight"`, or `"both"`. The default, `NULL`, produces no tracing output.

`pnts.col` character or numeric. Specifies the color of observations in a plot. The argument is used when `tracing` is not NULL. If `pnts.col = NULL`, observations will not be displayed.

`display` logical. Display a simple plot of results at the end of computations?

`pacific.region` logical (default is FALSE). When set to TRUE, indicates that the study area is crossed by the 180th meridian, such as being within the Pacific Ocean. In this case, the coordinates of `xy.dat` and `study.pol` are transformed to eliminate the artificial abrupt change in x-coordinate values at the 180th meridian.

`.do_recurse` Logical. Controls recursion (for internal use only). Default is `TRUE`. Normal users should not modify this parameter.

Details

The Algorithm: 1) Split-point Placement: The function places a predetermined number of split-points (`n.split.pts + 1`) along the perimeter of the study area (`study.pol`) or the convex hull of observation locations (`xy.dat`) if a study area polygon is not provided. These split-points are evenly spaced, resulting in a distance between points equal to 1/(n.split.pts + 1) fraction (1/16 by default) of a polygon circumference.

2) Straight Split-lines: Straight split-lines are generated by connecting the split-points. The total number of straight split-lines generated is equal to the value of `(n.split.pts + 1) * n.split.pts / 2` or to `choose(n.split.pts + 1, 2)`. This holds true only in the first iteration when `same.n.split` is set to `FALSE` or in all iterations when `same.n.split` is set to `TRUE`. Note that the total number of split-lines generated will not be equal to the number of split-lines evaluated since split-lines that

cross polygon boundary or do not pass sample size, area or location number subdivision criteria are not evaluated.

3) Subdivisions: Each split-line spatially divides the data and study area polygon into two subsets.

4) Criteria: Both subsets are then checked to see if they meet sample size, area and location number criteria.

5) Obtaining comparison values: Subsets that meet the criteria are compared using the `generalize.f` and `compare.f` functions to obtain a comparison value. First, each subset is passed to `generalize.f` to obtain a generalization value, such as the Pielou evenness index for the `"pielou"` method; see the **Note** section for clarification. These values are then used by `compare.f` to compare the subsets. In this way, each split-line that passed all subdivision criteria is assigned a comparison value.

6) Best Straight Split-line Selection: The best performing straight split-line is determined based on whether the `maximize` argument is set to `TRUE` or `FALSE`. If `maximize` is `TRUE`, the best split-line is the one with the highest comparison value; if `maximize` is `FALSE`, the best split-line has the lowest comparison value.

The combination of the `generalize.f`, `compare.f`, and `maximize` arguments can be provided, enabling the creation of custom methods, or it can be determined internally based on the chosen preset subdivision methods (see below).

7) Nonlinear Split-lines: If the best straight split-line meets the quality criteria specified by the `c.Q.crit`, it serves as the basis for generating variously shaped curves (nonlinear split-lines). These curves are produced using splines, which are mathematical functions that can create smooth and flexible curves.

To generate the curves, a number of knots (control points) for the splines are distributed evenly along a set of lines orthogonal to the best straight split-line. These orthogonal lines are also evenly distributed along the straight split-line itself. The number of knots and lines used can be adjusted through the parameters `c.Y.knots` and `c.X.knots`, respectively.

The algorithm then iterates through this network of knots, considering different combinations of knots, to produce curves. By varying the selection and arrangement of knots, different shapes of curves are generated.

8) Nonlinear Split-line Evaluation and Selection: Curves are then processed in the same manner as straight split-lines in steps 3 to 6.

9) Final Split-line Selection and Establishment of Subdivision: If the best curve outperforms the best straight split-line by a margin of `c.crit.improv`, the best split-line becomes nonlinear. Otherwise, it remains straight. The split-line must also satisfy the criteria established by the `Q.crit` argument to be used for subdivision.

10) Recursive Iteration: The process described above is iteratively applied, resulting in a collection of selected split-lines with their performance values. These split-lines hierarchically subdivide space and data, forming polygons of various shapes.

Preset Methods: Preset methods and their combinations of `compare.f`, `generalize.f`, and `maximize`:

`"sorensen"` Functions calculate the Sorensen similarity index (Sorensen 1948). `maximize = FALSE`. Values vary from 0 to 1.

`"pielou"` Functions calculate the mean proportional decrease in Pielou evenness (Pielou 1966) that occurs after polygon subdivision. `maximize = TRUE`. Values vary from 0 to 1.

`"morisita"` Functions calculate the Morisita overlap index (Morisita 1959). `maximize = FALSE`. Values vary from 0 to 1.

`"horn.morisita"` Functions calculate the Morisita overlap index, as modified by Horn (1966). `maximize = FALSE`. Values vary from 0 to 1.

All preset methods currently available are specifically designed for bioregionalization purposes. These methods require two key inputs: the coordinates of fossil taxon occurrences (`xy.dat`) and the names or IDs of taxa (`data`). These names or IDs should be structured as character or numeric vectors, with each element corresponding to a row in the `xy.dat` data frame. Each method compares fossil communities on opposite sides of a split-line, aiming to minimize similarity or maximize difference. The output yields biogeographical provinces with a hierarchical structure.

Value

A `hespdiv` object, which is a list with seven elements:

`poly.stats` A data frame containing information about polygons established by the selected split-lines. Its columns are:

- `rank`: The rank of a polygon. It corresponds to the rank of the split-line that produced the polygon and to the polygon's position in the hierarchical structure of the subdivision.
- `plot.id`: The ID assigned to the polygon. It corresponds to the order in which the polygon was processed during the `hespdiv` analysis.
- `root.id`: The ID of the parent polygon whose subdivision resulted in the current polygon.
- `n.splits`: The number of straight split-lines evaluated in an attempt to subdivide the polygon. This count excludes split-lines that crossed the polygon boundary or did not meet area, sample size, or location-number criteria.
- `n.obs`: The number of observations inside the polygon.
- `mean`: The average comparison value of the straight split-lines used in the attempted subdivision of the polygon. This value reflects the general spatial heterogeneity of the data.
- `sd`: The standard deviation of the comparison values of the straight split-lines used in the attempted subdivision. It indicates the extent of anisotropy or variation in spatial heterogeneity within the polygon.
- `str.best`: The comparison value of the best straight split-line produced within the polygon.
- `str.z.score`: The z-score of the comparison value of the best straight split-line within the polygon. It indicates how outstanding the best straight split-line is relative to other evaluated straight subdivisions.
- `has.split`: Logical. Indicates whether a subdivision was established in the polygon.
- `is.curve`: Logical. Indicates whether the established subdivision was obtained using a curve. If `has.split` is `FALSE`, this value is `NA`.
- `crv.best`: The same as `str.best`, but for nonlinear split-lines.
- `crv.z.score`: The same as `str.z.score`, but for nonlinear split-lines.
- `c.improv`: The improvement in comparison value achieved by using nonlinear split-lines instead of straight split-lines.

`split.stats` A data frame containing information about established split-lines. Its columns can be interpreted similarly to those in `poly.stats`, but from the perspective of split-lines rather than polygons. For example, `rank` is the rank of the split-line, not of the polygon. The

performance column contains the comparison value of the split-line and corresponds to either str.best or crv.best in poly.stats, depending on the value of is.curve. The same applies to the z.score column.

split.lines A list of data frames containing coordinates of established split-lines. The order of split-lines in this list corresponds to their order in split.stats.

polygons.xy A list of data frames containing coordinates of polygons established by the split-lines. The order of polygons in this list corresponds to their order in poly.stats.

poly.obj A list of polygon objects, that is, outputs of generalize.f for each polygon. The order of elements in this list corresponds to the row order of poly.stats and the polygon order in polygons.xy.

call.info Information about the hespdiv call, including the method and arguments used.

str.difs A list containing comparison values of evaluated straight split-lines for each polygon. The elements of this list correspond to the rows of poly.stats.

Note

Please note that if you use the method argument, the arguments generalize.f, compare.f, and maximize are determined internally and should not be provided. Therefore, you should only assign values to these arguments when using a custom method, not a predefined one. Additionally, you can ignore the generalize.f argument even when applying custom methods. If generalize.f is set to NULL (default), the data remains unchanged, as generalize.f acts as an identity function. Hence, generalize.f is only an optional argument that allows to omit the transformation or generalization step in compare.f function, simplifying it.

Both generalize.f and compare.f inherit environments from parent functions: hespdiv() and .spatial_div(). This allows them to use additional variables from those environments, such as xy.dat, samp.xy, id1, and id2.

There is a small possibility that a nonlinear split-line may cross a polygon boundary. If the result contains such a split-line, or if an error related to this issue occurs, slightly change one of the arguments, such as n.split.pts, and re-run hespdiv().

Author(s)

Liudas Daumantas

References

Horn, H. S. (1966). Measurement of" overlap" in comparative ecological studies. The American Naturalist, 100(914), 419-424.

Morisita, M. (1959). Measuring of interspecific association and similarity between assemblages. Mem Fac Sci Kyushu Univ Ser E Biol, 3, 65-80.

Pielou, E. C. (1966). The measurement of diversity in different types of biological collections. Journal of theoretical biology, 13, 131-144.

Sorensen, T. A. (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons. Biol. Skar., 5, 1-34.

Examples

```
# Simulated data with a known boundary at x = 0.45 to illustrate boundary detection

study.area <- data.frame(
  x = c(0, 0.8, 1, 0.6, 0, 0),
  y = c(0, 0, 0.4, 1.1, 1.1, 0)
)

set.seed(852)
# Simulate 100 occurrence coordinates
N <- 100
xy.dat_arg <- data.frame(
  x = rpois(N, 4) / 10,
  y = rpois(N, 4) / 10
)

xy.dat_arg <- xy.dat_arg[order(xy.dat_arg$x), ]

# Simulate a boundary at x = 0.45
n_left <- sum(xy.dat_arg$x < 0.45)

set.seed(1)
common_data <- letters[1:5]

left_data <- sample(
  c(common_data, LETTERS[1:10]),
  size = n_left,
  replace = TRUE,
  # common-endemic probability ratio 3:4
  prob = c(rep(3, 5), rep(4, 10))
)

right_data <- sample(
  c(common_data, LETTERS[11:20]),
  size = N - n_left,
  replace = TRUE,
  # common-endemic probability ratio 3:4
  prob = c(rep(3, 5), rep(4, 10))
)

data_arg <- c(left_data, right_data)

# Apply hespdiv
r <- hespdiv(
  data = data_arg,
  xy.dat = xy.dat_arg,
  n.split.pts = 6,  # small value used here for illustration
  method = "sor",   # subdivision minimizing Sorensen-Dice similarity
  S.crit = 0.3,     # minimum area size is 30% of study area
  study.pol = study.area,
  use.chull = FALSE
)
```

```
plot_hespdiv(r, n.loc = TRUE) + ggplot2::geom_vline(xintercept = 0.45)

# Detected split-line performance
r$split.stats$performance

# Sorensen-Dice similarity across the true simulated boundary
2 * length(intersect(left_data, right_data)) /
  (length(unique(left_data)) + length(unique(right_data)))
```

---

hsa *HespDiv Sensitivity Analysis*

Description

This function performs sensitivity analysis of the `hespdiv` method. Starting from a reference `hespdiv` object, it generates a specified number of alternative `hespdiv` calls by randomly sampling new values for selected arguments from user-provided sets.

Usage

```
hsa(
  obj,
  n.runs = 100,
  data.paired = TRUE,
  display = FALSE,
  images.path = NULL,
  pnts.col = 1,
  data = NULL,
  xy.dat = NULL,
  same.n.split = NULL,
  n.split.pts = NULL,
  N.crit = NULL,
  N.rel.crit = NULL,
  N.loc.crit = NULL,
  N.loc.rel.crit = NULL,
  S.crit = NULL,
  S.rel.crit = NULL,
  Q.crit = NULL,
  c.splits = NULL,
  c.Q.crit = NULL,
  c.crit.improv = NULL,
  c.X.knots = NULL,
  c.Y.knots = NULL,
  c.max.iter.no = NULL,
  c.fast.optim = NULL,
  c.corr.term = NULL,
  study.pol = NULL,
```

```
  use.chull = NULL,
  generalize.f = NULL,
  maximize = NULL,
  method = NULL,
  compare.f = NULL,
  .run.id = NULL,
  parallel = FALSE,
  RAM = NULL,
  load_prop = 0.8,
  chunk_size = workers * 2,
  workers = NULL,
  future_seed = TRUE
)
```

Arguments

obj	A hespdiv class object.
n.runs	Integer. The number of alternative hespdiv calls to evaluate.
data.paired	Logical. Controls whether alternative values of data are paired with corresponding elements of xy.dat.
display	Logical. Controls the value of the display argument in each hespdiv call.
images.path	Character or NULL. Path to an existing directory where PNG images of displayed results will be saved. If NULL (default), images are not saved.
pnts.col	Value passed to the pnts.col argument of each hespdiv call. Can be provided as atomic vector, vector or list of vectors.
data	A list of data objects (matrices, data frames, vectors, lists, or other supported data structures) used as alternative inputs for sensitivity analysis.
xy.dat, study.pol	Lists of data frames with two columns: x and y.
same.n.split, c.fast.optim, use.chull, c.splits	Logical vectors specifying alternative values for corresponding hespdiv arguments.
n.split.pts, c.max.iter.no, N.crit, N.loc.crit, c.X.knots, c.Y.knots	Integer vectors specifying alternative values for corresponding arguments.
N.rel.crit, N.loc.rel.crit, S.crit, S.rel.crit	Numeric vectors with values between 0 and 1.
Q.crit, c.Q.crit, c.crit.improv	Numeric vectors specifying alternative threshold or improvement criteria.
c.corr.term	Numeric vector with values between 0.01 and 0.2.
generalize.f, compare.f	Lists of functions defining custom similarity or generalization methods.
maximize	Logical vector of the same length as compare.f, indicating whether the corresponding similarity metric should be maximized.
method	Character vector specifying predefined similarity metrics.

.run.id	Integer. Runs with indices less than or equal to this value will be skipped. This can be used to resume an interrupted analysis.
parallel	Logical. If TRUE, runs are evaluated in parallel using **future.apply**. If FALSE (default), runs are evaluated sequentially.
RAM	Integer. Approximate amount of available RAM (in GB) used to limit the number of parallel workers when parallel = TRUE.
load_prop	Numeric in (0, 1]. Proportion of available CPU cores to use when determining the number of parallel workers automatically. Defaults to 0.8.
chunk_size	Integer. Number of runs submitted per batch (chunk) to the parallel backend when parallel = TRUE. Chunking does not change how many runs execute simultaneously (controlled by workers). Smaller values reduce the number of queued futures at once (often lowering RAM overhead). Defaults to workers*2.
workers	Integer. Number of parallel workers (CPU cores) to use when parallel = TRUE. At most this many hespdiv calls will be executed simultaneously. If NULL, the number of workers is determined automatically based on RAM and load_prop.
future_seed	Logical. Passed to future.apply::future_lapply(). If TRUE, RNG is managed by **future** to provide parallel-safe, reproducible random streams.

Details

Difference Between "hsa" and other sensitivity analysis functions: The hsa_detailed function evaluates all combinations of provided argument values, resulting in dense sampling of the parameter space at substantial computational cost. In contrast, hsa samples the parameter space stochastically and is generally more suitable for exploratory or large-scale sensitivity analyses.

In hsa_detailed, alternative argument values are provided as lists, whereas in hsa they are supplied as vectors or lists depending on the argument.

The hsa_sample_constrained function performs non-recursive hespdiv runs for each split-line produced based on different data subsamples. Thus, hsa is more general, as it allows to inspect sensitivity to other arguments.

Paired Arguments: When data.paired = TRUE, the same index is used to sample elements of data and xy.dat, allowing sensitivity analysis across datasets of differing size or composition. When FALSE, data and coordinates are sampled independently, enabling analyses based on noise addition or spatial shuffling.

Arguments defining custom methods (compare.f, generalize.f, maximize) are always treated as paired and must therefore have equal lengths.

Value

An object of class hsa, containing:

Alternatives A list of alternative hespdiv results produced during the sensitivity analysis.

Basis The reference hespdiv object whose arguments were perturbed.

Note

If a particular run produces a warning or an error, the corresponding list element will contain two components. In case of a warning, these are the resulting hespdiv object and the warning message. In case of an error, they are the arguments used for the call and the error message.

See Also

Other functions for hespdiv sensitivity analysis: change_base(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), plot_cs_hsa(), plot_hsa(), plot_hsa_q()

Other functions for hespdiv results post-processing: cross_comp(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), nulltest(), taxon_effect()

`hsa_detailed` *Detailed HespDiv Sensitivity Analysis*

Description

This function is one of the two that perform HespDiv sensitivity analysis. It creates and evaluates alternative hespdiv calls, according to the desired changes in method, data and other subdivision criteria arguments. As a result, it returns alternative hespdiv objects that can be directly compared with the original hespdiv object and with each other using `plot.hsa` and `hsa_quant` functions.

Usage

```
hsa_detailed(
  obj,
  comb.args = TRUE,
  pick.n.args = NULL,
  comb.type = NULL,
  n.combs = NULL,
  display = TRUE,
  images.path = NULL,
  paired = NULL,
  pnts.col = 1,
  data = NULL,
  xy.dat = NULL,
  same.n.split = NULL,
  n.split.pts = NULL,
  N.crit = NULL,
  N.rel.crit = NULL,
  N.loc.crit = NULL,
  N.loc.rel.crit = NULL,
  S.crit = NULL,
  S.rel.crit = NULL,
  Q.crit = NULL,
  c.splits = NULL,
  c.Q.crit = NULL,
  c.crit.improv = NULL,
  c.X.knots = NULL,
  c.Y.knots = NULL,
  c.max.iter.no = NULL,
  c.fast.optim = NULL,
```

```
  c.corr.term = NULL,
  study.pol = NULL,
  use.chull = NULL,
  generalize.f = NULL,
  maximize = NULL,
  method = NULL,
  compare.f = NULL
)
```

Arguments

obj	An object of hespdiv class. The base object whose call will be modified to produce alternative hespdiv objects.
comb.args	A Boolean value. Do you want to combine the provided argument values to make alternative hespdiv calls? If not, then at once only one argument will be modified, trying all provided values for it one by one.
pick.n.args	A numeric vector that controls how many arguments would you like to change at once in hespdiv runs. Multiple values allowed.
comb.type	A character determining how combinations of argument values are selected. Possible values: "all", "random", or "handpicked".
n.combs	An integer controlling how many argument value combinations should be randomly selected from all possible combinations when comb.type is "random".
display	A Boolean value. The value of the "display" argument in each hespdiv call.
images.path	A path to an existing directory where PNG images of the displayed results will be saved. If NULL (default), images won't be saved.
paired	Logical. Are the provided data and xy.dat arguments paired?
pnts.col	The value of the "pnts.col" argument in each hespdiv call.
data	A list containing matrices, time-series, lists, data frames, vectors, or other data structures.
xy.dat, study.pol	Lists of coordinate data frames. Each data frame must contain two columns named x and y.
same.n.split, c.fast.optim, use.chull, c.splits	Lists with a Boolean value (if used, should be different from the one in the basal hespdiv call).
n.split.pts, c.max.iter.no, N.crit, N.loc.crit, c.X.knots, c.Y.knots	Lists with integer values.
N.rel.crit, N.loc.rel.crit, S.crit, S.rel.crit	Lists with values between 0 and 1.
Q.crit, c.Q.crit, c.crit.improv	Lists of numeric values.
c.corr.term	A list of numeric values between 0.01 and 0.2.
generalize.f, compare.f	Lists of functions.
maximize	A list of logical values with the same length as the compare.f list.
method	A list of character values.

Details

Difference Between "hsa" And "hsa_detailed": The major difference between "hsa_detailed" and "hsa" is that the former produces all possible hespdiv calls from combinations of the provided hespdiv arguments. Therefore, it samples a much smaller segment of the parameter space but more densely, requiring much more computation time. Although such behavior may be desired in some cases, the "hsa" function is generally more suitable for performing hespdiv sensitivity analysis.

Additionally, alternative values for hespdiv arguments in the "hsa_detailed" function are provided in lists, whereas in the "hsa" function, they are provided in vectors or lists (depending on the argument).

Internally Set Default Argument Values: When comb.args is TRUE, the default value of comb.type is "all".

When comb.args is TRUE and pick.n.args is NULL (default), the value of pick.n.args will be changed to a vector 1:N, where N is the maximum possible value of pick.n.args. The maximum possible value for pick.n.args depends on the hespdiv arguments provided. Each hespdiv argument that influences the results is counted as one, except for "data" and "xy.dat" when paired is TRUE, and all four arguments ("method", "compare.f", "generalize.f", and "maximize") that define the subdivision method, as the pair/group of them is counted as one. Therefore, N can vary from 1 (single argument provided) to 22 (all arguments provided and paired is FALSE). If comb.args is FALSE, then pick.n.args should be NULL. Using pick.n.args = 1 is the same as setting comb.type to FALSE.

Paired Arguments: If paired is TRUE, the "data" and "xy.dat" elements with the same index are treated as one value of the same argument. Therefore, the provided lists of "data" and "xy.dat" should be of the same length. Pairing of "data" and "xy.dat" can be useful, for example, when you want to re-run hespdiv after adding or removing some observations (these changes should be made in both "xy.dat" and "data") to test how hespdiv results are influenced by some extra observations or the number of observations in general. When paired is FALSE, the number of observations in "data" and "xy.dat" must be the same as it was in the call of the base hespdiv object. This option allows you to re-run hespdiv after adding some noise to the object features (via changes in "data") or coordinates (via changes in "xy.dat") to test how hespdiv results are influenced by the data itself or localization.

By default, arguments determining the custom method ("compare.f", "generalize.f", "maximize") are paired, similar to how "data" and "xy.dat" are paired when paired is TRUE. Thus, the lists of "compare.f", "generalize.f", and "maximize" should be of the same length.

Value

An object of class `hsa`. The object is a list with two elements:

`Alternatives` A list containing the alternative `hespdiv` objects produced by the sensitivity analysis.

`Basis` The original `hespdiv` object whose call was modified to produce the alternative subdivisions.

Note

Use "pnts.col" of length >1 only when the number of observations does not change.

If a particular call produced a warning or error, then a list of length 2 will be returned for that call. If a warning was produced, then the first element of the list will hold the created hespdiv object, and the second element will contain the warning message. In the case of an error, the first element will be a list of arguments used to produce the call, and the second element will contain the error message.

Author(s)

Liudas Daumantas

See Also

Other functions for hespdiv sensitivity analysis: `change_base()`, `hsa()`, `hsa_quant()`, `hsa_sample_constrained()`, `plot_cs_hsa()`, `plot_hsa()`, `plot_hsa_q()`

Other functions for hespdiv results post-processing: `cross_comp()`, `hsa()`, `hsa_quant()`, `hsa_sample_constrained()`, `nulltest()`, `taxon_effect()`

`hsa_quant` *Quantify the stability of hespdiv clusters*

Description

This function evaluates the stability of the basal subdivision clusters using hespdiv sensitivity analysis results. It does so by calculating Jaccard similarities between the observations of basal subdivision clusters and the observations of alternative subdivision clusters. For each basal cluster, the function identifies the most similar analog cluster within each alternative subdivision. The stability of each basal cluster can be assessed by examining the distribution of similarity values with their corresponding analog clusters. If a highly similar cluster reappears in multiple alternative subdivisions, it indicates that the basal cluster is stable.

Usage

```
hsa_quant(obj, probs = c(0.05, 0.5, 0.95))
```

Arguments

`obj`	An object of class `hsa`.
`probs`	A numeric vector of probabilities with values in the range $0 \leq p \leq 1$. This argument is used to calculate quantiles of Jaccard similarity values.

Details

If a base subdivision cluster obtains a distribution of high similarity values, it is considered stable and existing. Low analog-cluster similarity values may indicate that a base cluster is an artifact of the `hespdiv()` computation.

The more technical description of how `hsa_quant` works:

Obtaining alternative hespdiv clusters: The function filters the `xy.dat` coordinates of the basal subdivision using all the polygons of alternative subdivisions, obtaining alternative hespdiv clusters.

Quantifying Jaccard similarity: The function measures the Jaccard overlap index between the observations of the basal subdivision clusters and the observations of the alternative clusters.

Identification of analog clusters and value assignments: Each basal hespdiv cluster from each alternative subdivision is assigned the ID of the cluster that produced the maximum Jaccard similarity value, along with the corresponding similarity value.

The purpose of the `hsa_quant` function is to address situations where hespdiv polygons, despite having different geometry and location, may filter nearly identical sets of observations, leading to similar hespdiv clusters. This can occur when the spatial coverage of observations is incomplete and irregular, or when the boundaries between hespdiv polygons are expected to be open, soft, or fuzzy, such as in the case of boundaries between bioregions. In such cases, visual hespdiv sensitivity analysis alone may show irregular and non-converging distributions of split-lines. However, `hsa_quant` can reveal that these irregular polygons are based on nearly identical clusters of observations, indicating a strong spatial structure within the analyzed data. Conversely, if the observations within these clusters significantly differ, it indicates that the basal clusters are specific to the hespdiv parameters used and likely lack ontological meaning.

Thus, by analyzing the similarity values between clusters of observations, `hsa_quant` facilitates the assessment of the stability and reliability of basal subdivision clusters, aiding in evaluating their significance.

Value

A list containing three data frames:

jaccard.quantiles Quantiles of Jaccard similarities between the basal cluster and the analog clusters from alternative subdivisions.

jaccard.similarity Jaccard similarity values between the basal cluster and the analog cluster from each alternative subdivision.

analog.clusters IDs of the hespdiv polygons that produced the analog clusters in each alternative subdivision.

Note

You can use the hsa_quant function to track the evolution of hespdiv subdivisions over time by providing correctly formatted input. For instance, you can obtain the basal subdivision for time bin 1 using the hespdiv function. Then, using the hsa function, you can specify the paired xy.dat and data from time bin 2. The resulting hsa object can be inputted into `hsa_quant` The `hsa_quant` result will then provide insights into extinctions, speciations, fusions, and splits of hespdiv polygons/clusters that occur between time bin 1 and 2. This allows for the analysis of changes and dynamics in hespdiv subdivisions over time.

Author(s)

Liudas Daumantas

See Also

Other functions for hespdiv sensitivity analysis: change_base(), hsa(), hsa_detailed(), hsa_sample_constrained(), plot_cs_hsa(), plot_hsa(), plot_hsa_q()

Other functions to evaluate hesdpiv cluster stability: plot_hsa_q()

Other functions for hespdiv results post-processing: cross_comp(), hsa(), hsa_detailed(), hsa_sample_constrained(), nulltest(), taxon_effect()

---

hsa_sample_constrained

Constrained HespDiv Sensitivity Analysis by Subsampling

Description

Conduct a constrained sensitivity analysis on a hespdiv object by repeatedly subsampling observations within each polygon. Each subsample is used to call hespdiv with recursion disabled (i.e., single-split only).

Usage

```
hsa_sample_constrained(
  obj,
  n.runs = 100,
  subsample_factor = 0.7,
  RAM = NULL,
  load_prop = NULL,
  chunk_size = 8,
  workers = NULL
)
```

Arguments

obj	A hespdiv object.
n.runs	Integer. The number of subsampling runs to perform (default: 100).
subsample_factor	Numeric proportion of data to subsample within each polygon (0 to 1]. For example, 0.7 means 70% of the data in each polygon are retained.
RAM	Integer. Approximate amount of RAM in GB to guide how many parallel workers to use. The function uses up to 80\ but also caps the number of workers at RAM.

load_prop	Numeric value (0,1]. Specifies the proportion of available CPU cores or RAM to be used for setting up parallel workers. For example, load_prop = 0.8 uses 80% of the available resources. If both load_prop and RAM are provided, the number of workers will be the minimum based on the constraints imposed by both. Defaults to 0.8 if not provided.
chunk_size	Integer. Number of runs submitted per batch. Parallelism is controlled by workers; chunking mainly controls memory/queue size.
workers	A number of parrallel workers. Determines the number of hespdiv runs to be processed in parallel.

Details

For each polygon in the hespdiv object, this function draws subsample_factor of the data (by default 70%), creating multiple random subsamples (n.runs). These are processed in chunks (as given by chunk_size) and runs in each chunk being parallelized to manage memory usage.

Value

A hsa_constrained class object, a list with two elements:

- **Alternatives** A named list corresponding to each polygon where each entry is another list of hespdiv results for each subsample run.
- **Basis** The original hespdiv object (obj).

See Also

hespdiv for details on the main function. **future.apply**

Other functions for hespdiv sensitivity analysis: change_base(), hsa(), hsa_detailed(), hsa_quant(), plot_cs_hsa(), plot_hsa(), plot_hsa_q()

Other functions for hespdiv results post-processing: cross_comp(), hsa(), hsa_detailed(), hsa_quant(), nulltest(), taxon_effect()

---

nulltest *Test significance of split-lines in spatial subdivision*

Description

Assess the statistical significance of each split-line (bioregion boundary) identified by hespdiv by comparing its observed performance to a null distribution. The null is generated by permuting the data n times and recomputing the split-line performance after each permutation. Multiple shuffling strategies are supported to probe the influence of spatial structure on delineation.

Usage

```
nulltest(
  obj,
  n = 999,
  maintain.n = TRUE,
  shuffle.scope = "within",
  shuffle.type = "localities"
)
```

Arguments

obj	An object of class hespdiv.
n	Integer. Number of permutations used to form the null distribution.
maintain.n	Logical. Only honored when shuffle.scope = "within" and shuffle.type = "localities". If TRUE, attempts to keep the randomized child polygon occurrence counts close to the observed (maximum discrepancy up to max_n/2, where max_n is the largest locality size). Ignored otherwise.
shuffle.scope	Character. Either "all" (shuffle across the full study area) or "within" (shuffle only within each parent polygon).
shuffle.type	Character. Either "localities" (shuffle whole localities, preserving assemblages) or "occurrences" (shuffle individual occurrences). "localities" is generally preferred.

Details

Two shuffling scopes are available:

"all": Shuffle across the entire study area, ignoring polygonal subdivisions.

"within": Shuffle only within each parent polygon (the region in which the split-line is nested), preserving local spatial structure.

Two shuffling types are available:

"localities": Shuffle whole localities, preserving each site's assemblage (recommended, since occurrences within a locality are not independent).

"occurrences": Shuffle individual occurrences across sites (use with caution; may violate within-locality independence).

For each split-line, the function reports the observed performance, the mean and standard deviation of the permuted (null) performances, an empirical one-sided p-value (proportion of permuted values *as or more extreme* than observed; ties included), and a z-score quantifying departure from the null.

Value

Invisibly returns an object of class nullhespdiv, a list with:

- $stats, a data frame summarizing each split-line with:
  - performance: observed performance.

  - `mean.random`: mean of null performances.
  - `sd.random`: standard deviation of null performances.
  - `p.val`: empirical one-sided p-value (ties included).
  - `z.score.random`: standardized effect size.
- `$null`, a matrix or data frame containing all null performance values for every split-line across permutations.

See Also

Other functions for hespdiv results post-processing: `cross_comp()`, `hsa()`, `hsa_detailed()`, `hsa_quant()`, `hsa_sample_constrained()`, `taxon_effect()`

Examples

```
# if split-line is strongly significant, the choice of parameters should not
# matter. For example (look at p-value, z.score.random, sd.random and
# mean.random):
(nulltest(example_hespdiv, maintain.n = FALSE, shuffle.type = "occurrences"))
(nulltest(example_hespdiv, maintain.n = FALSE, shuffle.type = "localities"))
(nulltest(example_hespdiv, maintain.n = TRUE, shuffle.type = "localities"))
```

`plot.nullhespdiv` *Plot a nullhespdiv object*

Description

Plot method for a `nullhespdiv` object.

Usage

```
## S3 method for class 'nullhespdiv'
plot(x, ...)
```

Arguments

`x`	A `nullhespdiv` object.
`...`	Additional arguments passed to `graphics::boxplot()`.

Value

No return value. Called for the side effect of plotting a `nullhespdiv` object.

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot_cs_hsa(), plot_hespdiv(), plot_hsa(), plot_hsa_q(), poly_scheme(), polypop()

Examples

```
test_results <- nulltest(example_hespdiv)
plot(test_results)
```

---

plot_cs_hsa *Visualize Constrained HespDiv Sensitivity Analysis Results*

Description

Displays the alternative (subsampled) hespdiv subdivisions and the basal (original) hespdiv subdivision on one or multiple plots, illustrating how the split-lines vary across different ranks. Additionally, for each alternative split-line (which is defined by a start and end coordinate), the function aggregates identical endpoints and overlays their counts on the plot.

Usage

```
plot_cs_hsa(
  obj,
  type = 1,
  rank = NULL,
  col_basal = "gray20",
  main,
  col_boundary = 7,
  col_alternatives = "lightyellow3",
  max_lwd = 2.5,
  min_lwd = 0.75,
  alpha_alt = 0.6
)
```

Arguments

obj	An object of class hsa_constrained, the output of hsa_sample_constrained.
type	An integer indicating the type of plot. Defaults to 1. 1 A single plot overlaying all alternative lines and the basal lines, colored or width-coded by rank. 2 Multiple plots, each corresponding to a specific rank. Alternative lines are displayed in a user-defined color (default "lightyellow3"), with the basal line highlighted in another color.
rank	Integer. Optional. When type = 2, if provided the function will only generate plots for the specified rank. If NULL (default), plots for all unique ranks will be produced.

col_basal	Character or numeric specifying the color of basal split-lines (default "gray20").
main	Character. Title for the plot(s).
col_boundary	Character or numeric specifying the color of the outer (first) polygon boundary (default 7).
col_alternatives	Character or numeric specifying the color of alternative split-lines (default "lightyellow3").
max_lwd	Numeric. The maximum line width for the highest-ranked split-line (default 2.5).
min_lwd	Numeric. The minimum line width for the lowest-ranked split-line (default 0.75).
alpha_alt	Numeric in the range [0, 1]. The transparency of alternative split-lines (default 0.6).

Details

- In type = 1, the function creates a single plot showing all alternative split-lines overlaid on the first polygon boundary, plus all basal split-lines of the hespdiv basis object.
- In type = 2, the function creates separate plots, each focusing on polygons of a specific rank, drawing alternative lines in the user-specified color (with transparency) and the basal line in another color or line width. If a specific rank is provided, only that rank is plotted.
- In both cases, after drawing the alternative split-lines the function aggregates their endpoints (start and end coordinates) and overlays the count at each unique coordinate using text().

Value

NULL. The function is called for its side effect of generating one or more plots.

See Also

Other functions for hespdiv sensitivity analysis: change_base(), hsa(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), plot_hsa(), plot_hsa_q()

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot.nullhespdiv(), plot_hespdiv(), plot_hsa(), plot_hsa_q(), poly_scheme(), polypop()

---

plot_hespdiv *Plot hespdiv results*

Description

This function is used to plot the results obtained with the hespidv function. The plot showcases subdivisions of the study area by split-lines, visualizing their performances or rank with colors or line widths. Additionally, it can display the spatial distribution of observations and number of observations in each location.

Usage

```
plot_hespdiv(
  obj,
  type = "color",
  n.loc = FALSE,
  performance = TRUE,
  legend_title = NULL,
  title = NULL,
  subtitle = NULL,
  pnts.col = NULL,
  seed = 10
)
```

Arguments

obj	A hespdiv object.
type	A character. Either "width" or "color" (default "color"). Determines whether quality of split-lines is expressed by line width or color.
n.loc	A Boolean value. Would you like to visualize the number of observations at each location? Only possible, when there are localities with more than one observation. If the type is 'color,' the number of observations is expressed through point sizes. Otherwise, they are expressed using color in a logarithmic scale.
performance	logical. TRUE - display split-line performance, FALSE - rank. Displaying rank makes the spatial dendrogram clearer.
legend_title	A character value that indicates the title of the legend for the split-lines. The default is built according to the method information available in "obj$call.info".
title	A character that indicates the title of the plot.
subtitle	A character that indicates the subtitle of the plot.
pnts.col	A character or numeric vector providing color codes for data points.
seed	An integer value that indicates seed used to randomize the colors of the split-lines. Only meaningful, when argument `type = "width"`. Try setting a different value, if colors of parallel split-lines or nearby labels look too similar or to increase the general appeal of the graph.

Details

The return ggplot object can be edited as any other ggplot objects by removing undesired elements, changing theme or overlying the plot with additional elements.

Value

A ggplot object.

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot.nullhespdiv(), plot_cs_hsa(), plot_hsa(), plot_hsa_q(), poly_scheme(), polypop()

Examples

```
plot_hespdiv(example_hespdiv)
plot_hespdiv(example_hespdiv, type = "width")
plot_hespdiv(example_hespdiv, n.loc = TRUE)
```

---

plot_hsa *Visualize the stability of hespdiv polygons*

Description

The function uses hespdiv sensitivity analysis results to visually demonstrate the stability of the basal hespdiv subdivision. This is achieved by displaying both alternative and basal hespdiv subdivisions on the same plot.

Usage

```
plot_hsa(
  obj,
  alpha = 0.6,
  split.col = "gray20",
  pnts.col = NULL,
  pol.col = "7",
  type = 1,
  basal.col = 2,
  max.lwd = 3,
  min.lwd = 0.5,
  split.col.seed = NULL,
  newplot = TRUE,
  seperated = TRUE
)
```

Arguments

obj	An object of class hsa
alpha	The alpha value for transparency of split lines. Default is 0.6.
split.col	Color of alternative subdivision split-lines. Default is "gray20".
pnts.col	The color of data points. Default is NULL.
pol.col	The color of polygons. Default is "7".
type	An integer indicating the type of plot. Default is 1.
basal.col	The color of basal subdivision split-lines.

max.lwd	The maximum line width for split-lines. Default is 3.
min.lwd	The minimum line width for split-lines. Default is 0.5.
split.col.seed	A seed for generating random colors for split lines. Default is NULL.
newplot	Create a plot in new device?
seperated	Boolean. When type is >= 3, open a new graphical device for each rank?

Details

The type parameter determines the type of plot generated:

1 Basic plot: displays the alternative and basal hespdiv subdivisions on the same plot without split-line ranks or titles.

2 Plot with split-line ranks: includes split-line ranks in the plot. Each split-line is assigned a different line width based on its rank.

3 Plot with separate ranks: generates multiple plots, each representing split-line ranks up to a certain value.

4 Plot with separate and isolated ranks: similar to mode 3 but isolates split-line ranks. Generates multiple plots, each representing a specific split-line rank.

If the alternative subdivisions spatially converge but the basal subdivision lies far from the zone of convergence, you can use change_base to select a more representative alternative subdivision to serve as the basal subdivision. However, you should verify that the arguments used in that subdivision are appropriate.

Value

No return value, called for plotting sensitivity analysis results.

Note

newplot allows the legend to be rendered correctly in types 2 and 3, and helps with line rendering in general when drawing in an active device (use broom otherwise to delete devices).

Author(s)

Liudas Daumantas

See Also

Other functions for hespdiv sensitivity analysis: change_base(), hsa(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), plot_cs_hsa(), plot_hsa_q()

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot.nullhespdiv(), plot_cs_hsa(), plot_hespdiv(), plot_hsa_q(), poly_scheme(), polypop()

plot_hsa_q *Plot the stability of hespdiv clusters*

Description

This function visualizes the stability of basal subdivision clusters obtained from hsa_quant

Usage

```
plot_hsa_q(obj, hist = FALSE)
```

Arguments

obj	The output of hsa_quant (an object of class hsa_quant).
hist	A Boolean value. If FALSE, EPDFs obtained with plot(density(jac.sim)) will be displayed instead of histograms.

Details

The stability of each basal cluster is revealed by the distribution of Jaccard similarity values with the 'analogues' clusters found in alternative hespdiv subdivisions. For example, a unimodal distribution with a peak at high similarity values (>0.8) indicates that the basal hespdiv cluster is stable, even if the polygon boundaries are not. This situation may arise when there is indeed a spatial structure within the data, but there are also wide gaps between sampled regions (or more generally when there is limited spatial data coverage). A unimodal distribution with a peak at medium values (0.4-0.6) and a tail to higher values could also indicate a more persistent spatial structure. On the other hand, a single peak at low values (<0.4) indicates low cluster stability (e.g., bioregion does not exist). Finally, uniform, bimodal, or other more complex distributions may indicate that the stability and existence of the corresponding basal cluster depend on the parameters used in alternative hespdiv calls.

Value

None

Author(s)

Liudas Daumantas

See Also

Other functions for hespdiv sensitivity analysis: change_base(), hsa(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), plot_cs_hsa(), plot_hsa()

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot.nullhespdiv(), plot_cs_hsa(), plot_hespdiv(), plot_hsa(), poly_scheme(), polypop()

Other functions to evaluate hesdpiv cluster stability: hsa_quant()

polypop *Remove polygons from rgl device*

Description

This function allows you to interactively select and remove unwanted polygons from a 3D plot created with the blok3d function.

Usage

```
polypop(obj, height)
```

Arguments

obj	The hespdiv object used to create the currently active rgl device with the blok3d function.
height	A character value that indicates the height co-ordinate.

Value

No return value. Called for the interactive modification of a plot created by blok3d

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot.nullhespdiv(), plot_cs_hsa(), plot_hespdiv(), plot_hsa(), plot_hsa_q(), poly_scheme()

poly_scheme *Schematic plot of hespdiv polygons*

Description

This function generates a schematic visualization of subdivided territory. It highlights the location of each polygon by displaying their centroids, ID labels, and punctuated lines that connect the polygon centroids with the split-lines that created them. This visualization represents the spatial arrangement of hespdiv polygons within the territory in 2D.

Usage

```
poly_scheme(obj, segment = TRUE, id = TRUE, seed = 1)
```

Arguments

obj	A hespdiv object.
segment	A Boolean value. Display the punctuated lines joining the polygon centroids with the split-lines?
id	A Boolean value. Display the IDs of polygons?
seed	An integer value that determines the random set of colors used in visualization of split-lines and polygons.

Value

A ggplot object.

Note

A much clearer way to visualize polygons is by using the blok3D function, with height = "rank". However, a 3D plot is less suitable option for papers.

Author(s)

Liudas Daumantas

See Also

Other HespDiv visualization options: blok3d(), create_gif(), dendro(), plot.nullhespdiv(), plot_cs_hsa(), plot_hespdiv(), plot_hsa(), plot_hsa_q(), polypop()

Examples

```
poly_scheme(example_hespdiv)
```

---

print.hespdiv *Print a hespdiv object*

Description

Formats and prints the rounded split.stats data frame from a hespdiv object.

Usage

```
## S3 method for class 'hespdiv'
print(x, ...)
```

Arguments

x	A hespdiv object.
...	Additional arguments, currently ignored.

Value

Invisibly returns `x`.

Author(s)

Liudas Daumantas

`print.nullhespdiv` *Print the results of nullhespdiv object*

Description

print method for nullhespdiv class object

Usage

```
## S3 method for class 'nullhespdiv'
print(x, ...)
```

Arguments

`x`	A nullhespdiv class object
`...`	other arguments

Value

x

Author(s)

Liudas Daumantas

`remove_splits` *Remove split-lines*

Description

Function returns hespdiv object, without split-lines of specified id.

Usage

```
remove_splits(obj, split.id, depend.splits = TRUE)
```

Arguments

obj	hespdiv object
split.id	vector of split-line ids
depend.splits	logical. Remove split-lines that depend on specified split-lines? If FALSE, only end-nodes of spatial dendrogram are removed.

Value

hespdiv object

Examples

```
if (requireNamespace("HDData")) {

  # Inspect the hespdiv object
  print(plot_hespdiv(HDData::hd))

  # Remove weak split-lines
  weak_splits <- which(HDData::hd$split.stats$performance >= 0.3)
  performance_filtered <- remove_splits(obj = HDData::hd, split.id = weak_splits)
  print(plot_hespdiv(performance_filtered))

  # Remove non-significant split-lines
  plot(HDData::nl)
  nsig_splits <- which(HDData::nl[[1]]$quantile >= 0.05)
  sig_filtered <- remove_splits(obj = HDData::hd, split.id = nsig_splits)
  print(plot_hespdiv(sig_filtered))

  # Remove only if a split-line has no dependent split-lines
  unchanged_hd <- remove_splits(obj = HDData::hd, split.id = 4, depend.splits = FALSE)
  print(plot_hespdiv(unchanged_hd))

  # Remove the split-lines indicated as well as all other split-lines
  # that structurally depend on them (default behavior)
  changed_hd <- remove_splits(obj = HDData::hd, split.id = 4, depend.splits = TRUE)
  print(plot_hespdiv(changed_hd))
}
```

---

taxon_effect *Taxon-level leave-out contributions for split-lines*

Description

For each unique taxon label in `obj$call.info$Call_ARGS$data` (for example, species or genus), remove all of its occurrences from both child polygons of every split, recompute split-line performance, and record both the resulting value and its difference from the original.

Usage

```
taxon_effect(obj)
```

Arguments

obj	A hespdiv object.

Details

Let P be the split-line performance stored in obj$split.stats$performance. For a focal taxon t, we compute P^{-t} by removing all occurrences of t. We report Δ so that $\Delta > 0$ always indicates a **positive contributor** (that is, removal worsens performance):

- If obj$call.info$Call_ARGS$maximize = FALSE (lower is better, for example similarity), $\Delta = P^{-t} - P$.
- If obj$call.info$Call_ARGS$maximize = TRUE (higher is better, for example distance), $\Delta = P - P^{-t}$.

If a taxon is absent from a split's parent polygons, elimination is a no-op and $\Delta = NA$.

Value

A list of class taxon_effect_result with:

elim.comp.vals Performance after removing each taxon; dimension [n_splits x n_taxa].

delta Signed contribution Δ as defined above; dimension [n_splits x n_taxa].

n_per_pol Counts of the focal taxon per split and polygon.

baseline The original performance vector.

See Also

Other functions for hespdiv results post-processing: cross_comp(), hsa(), hsa_detailed(), hsa_quant(), hsa_sample_constrained(), nulltest()

Index