Package: diffval (via r-universe)

October 20, 2026

Title Vegetation Patterns

Version 1.2.0.9001

Description Find, visualize and explore patterns of differential taxa in vegetation data (namely in a phytosociological table), using the Differential Value (DiffVal). Patterns are searched through mathematical optimization algorithms. Ultimately, Total Differential Value (TDV) optimization aims at obtaining classifications of vegetation data based on differential taxa, as in the traditional geobotanical approach (Monteiro-Henriques 2025, <doi:10.3897/VCS.140466>). The Gurobi optimizer, as well as the R package 'gurobi', can be installed from <https://www.gurobi.com/products/gurobi-optimizer/>. The useful vignette Gurobi Installation Guide, from package 'prioritizr', can be found here: <https://prioritizr.net/articles/gurobi_installation_guide.html>.

License GPL (>= 3)

URL https://point-veg.gitlab.io/diffval/

BugReports https://gitlab.com/point-veg/diffval/-/issues

Depends R (>= 2.10)

Imports graphics, parallel, stats

Suggests gurobi, utils

Encoding UTF-8

Language en-GB

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

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

Date/Publication 2025-11-24 12:30:47 UTC

RemoteUrl https://gitlab.com/point-veg/diffval

RemoteRef HEAD

RemoteSha 09b37ac7e2defda0903a8aedaa56fb0be37cc1f3

Contents

bigdata_tdv 2
explore_tabulation 4
identical_partition 5
internal_assignment 6
optim_tdv_gurobi_k_2 8
optim_tdv_hill_climb 9
optim_tdv_simul_anne 13
partition_tdv_grasp 17
partition_tdv_grdtp 18
tabulation 19
taxus_bin 21
tdv 22

Index **25**

`bigdata_tdv` *The Total Differential Value of a big phytosociological data set*

Description

Given a big phytosociological data set represented as a list, and a partition of the relevés in that list, this function calculates the respective Total Differential Value (TDV).

Usage

```
bigdata_tdv(
  phyto_list,
  p,
  n_rel,
  output_type = "normal",
  parallel = FALSE,
  mc_cores = getOption("mc.cores", 2L)
)
```

Arguments

`phyto_list`	A list. This is a very light representation of what could be a usual phytosociological table, registering only taxa presences. Each component should uniquely represent a taxon and should contain a vector (of numeric values) with the relevé(s) id(s) where that taxon was observed. Relevé's ids are expected to be represented by consecutive integers, starting with 1. The components of the list might be named (e.g. using the taxon name) or empty (decreasing further memory burden). However, for `output_type == "normal"` taxa names are useful for output interpretation.

p	A vector of integer numbers with the partition of the relevés (i.e., a k-partition, consisting in a vector with values from 1 to k, with length equal to the number of relevés in phyto_list, ascribing each relevé to one of the k groups).
n_rel	The number of relevés in phyto_list, obtained, for example, using the instruction length(unique(unlist(phyto_list))).
output_type	A character determining the amount of information returned by the function and also the amount of pre-validations. Possible values are "normal" (the default) and "fast".
parallel	Logical. Should function parallel::mclapply()) be used to improve computation time by forking? Not available on Windows. Refer to that function manual for more information. Defaults to FALSE.
mc_cores	The number of cores to be passed to parallel::mclapply() if parallel = TRUE. See parallel::mclapply() for more information.

Details

This function accepts a list (phyto_list) representing a phytosociological data set, as well as a k-partition of its relevés (p), returning the corresponding TDV (see tdv() for an explanation on TDV). Partition p gives the group to which each relevé is ascribed, by increasing order of relevé id. Big phytosociological tables can occupy a significant amount of computer memory, which mostly relate to the fact that the absences (usually more frequent than presences) are also recorded in memory. The use of a list, focusing only on presences, reduces significantly the amount of needed memory to store all the information that a phytosociological table contains and also the computation time of TDV, allowing computations for big data sets.

Value

If output_type = "normal" (the default) pre-validations are done (which can take some time) and a list is returned, with the following components (see tdv() for the mathematical notation):

ifp A matrix with the $\frac{a}{b}$ values for each taxon in each group, for short called the 'inner frequency of presences'.

ofda A matrix with the $\frac{c}{d}$ values for each taxon in each group, for short called the 'outer frequency of differentiating absences'.

e A vector with the e values for each taxon, i.e., the number of groups containing that taxon.

diffval A matrix with the $DiffVal$ for each taxon.

tdv A numeric with the TDV of matrix m_bin, given the partition p.

If output_type = "fast", only TDV is returned and no pre-validations are done.

Author(s)

Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Creating a group partition, as the one presented in the original article of
# the data set
groups <- rep(c(1, 2, 3), c(3, 11, 19))

# Removing taxa occurring in only one relevé, in order to reproduce exactly
# the example in the original article of the data set
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]

# Calculating TDV using tdv()
tdv(taxus_bin_wmt, groups)$tdv

# Converting from the phytosociologic matrix format to the list format
taxus_phyto_list <- apply(taxus_bin_wmt, 1, function(x) which(as.logical(x)))

# Getting the number of relevés in the list
n_rel <- length(unique(unlist(taxus_phyto_list)))

# Calculating TDV using bigdata_tdv(), even if this is not a big matrix
bigdata_tdv(
  phyto_list = taxus_phyto_list,
  p = groups,
  n_rel = n_rel,
  output_type = "normal"
)$tdv
```

`explore_tabulation` *Interactively explore a tabulation of a phytosociological matrix*

Description

This function plots an interactive image of a tabulation.

Usage

```
explore_tabulation(tab, palette = "Vik")
```

Arguments

`tab`	A list as returned by the `tabulation()` function.
`palette`	A character with the name of the colour palette (one of `grDevices::hcl.pals()` to be passed to `grDevices::hcl.colors()`. Defaults to "Vik".

Details

The function explore.tabulation accepts an object returned by the `tabulation()` function, plotting a condensed image of the respective tabulated matrix, permitting the user to click on the coloured blocks and receive the respective list of taxa names on the console.

Value

Returns invisibly, although it prints taxa names on the console upon the user click on the figure.

Author(s)

Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)
# Creating a group partition, as presented in the original article of
# the data set
groups <- rep(c(1, 2, 3), c(3, 11, 19))

# Removing taxa occurring in only one relevé in order to
# reproduce exactly the example in the original article of the data set
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]

# Sorts the phytosociological table, putting exclusive taxa at the top and
# plots an image of it
tabul <- tabulation(
  m_bin = taxus_bin_wmt,
  p = groups,
  taxa_names = rownames(taxus_bin_wmt),
  plot_im = "normal",
  palette = "Zissou 1"
)

# This creates an interactive plot (where you can click)
if (interactive()) {
  explore_tabulation(tabul, palette = "Zissou 1")
}
```

---

identical_partition *Do the vectors represent the same k-partition?*

Description

Checks if two vectors represent the same k-partition.

Usage

```
identical_partition(p1, p2)
```

Arguments

p1	A vector of integers representing a k-partition (taking values from 1 to k), of the same length of p2.
p2	A vector of integers representing a k-partition (taking values from 1 to k), of the same length of p1.

Details

Parameters p1and p2are vectors indicating group membership. In this package context, these vectors have as many elements as the columns of a phytosociological table, indicating the group membership of each relevé to one of k groups (i.e., a k-partition). This function checks if the two given vectors p1and p2 correspond, in practice, to the same k-partition, i.e., if the relevé groups are actually the same, but the group numbers are somehow swapped.

Value

TRUE if p1and p2 represent the same k-partitions; FALSE otherwise.

Author(s)

Tiago Monteiro-Henriques and Jorge Orestes Cerdeira. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Creating three 2-partitions
par1 <- c(1, 1, 2, 2, 2)
par2 <- c(2, 2, 1, 1, 1)
par3 <- c(1, 1, 1, 2, 2)

# Is it the same partition?
identical_partition(par1, par2) # TRUE
identical_partition(par1, par3) # FALSE
identical_partition(par2, par3) # FALSE
```

---

internal_assignment *Check the internal assignment of a given classification*

Description

Given a phytosociological table and a partition of its columns, this function checks the internal assignment of relevés to groups, based on the presence of taxa that are exclusive to each group (or group combination) as defined by the partition, distinguishing relevés assigned unambiguously to their group from those for which there is ambiguity.

Usage

```
internal_assignment(m_bin, p)
```

Arguments

m_bin	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
p	A vector of integer numbers with the partition of the relevés (i.e., a k-partition, consisting in a vector with values from 1 to k, with length equal to the number of columns of m_bin, ascribing each relevé to one of the k groups).

Details

The function accepts a phytosociological table (m_bin) and a k-partition of its columns (p), and assesses which relevés are assigned unambiguously to their group and which are not. The assignment of a relevé to a group is considered unambiguous when transferring it to another group would alter the pattern of differential taxa defined by p. Conversely, if a relevé could be moved to a different group without changing this pattern, its assignment is considered ambiguous.

Value

A list with the following components:

rel_ambiguous_assign A vector containing the names of the relevés with ambiguous assignment.

possible_assignments A data frame with all the possible assignments for the ambiguously assigned relevés.

iap The internal assignment precision (IAP), i.e., the proportion of relevés with unambiguous assignment.

iaa The internal assignment ambiguity (IAA), i.e., the proportion of relevés with ambiguous assignment (IAA = 1 - IAP).

Author(s)

Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Creating some group partitions
groups1 <- rep(c(1, 2, 3), c(3, 11, 19))
set.seed(1)
groups2 <- sample(rep(c(1, 2, 3), c(3, 11, 19)))

# In this case, all relevés are unambiguously assigned to a group
internal_assignment(taxus_bin, groups1)

# In this other case, some relevés could be moved to a different group, as
```

```
# their assignment is ambiguous
internal_assignment(taxus_bin, groups2)
```

`optim_tdv_gurobi_k_2` *Total Differential Value optimization using Gurobi*

Description

Given a phytosociological matrix, this function finds a partition in two groups of the matrix columns, which maximizes the Total Differential Value (TDV).

Usage

```
optim_tdv_gurobi_k_2(m_bin, formulation = "t-dependent", time_limit = 5)
```

Arguments

`m_bin`	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
`formulation`	A character selecting which formulation to use. Possible values are "t-dependent" (the default) or "t-independent". See Details.
`time_limit`	A numeric ("double") with the time limit (in seconds) to be passed as a parameter to Gurobi, Defaults to 5 seconds, but see Details.

Details

Given a phytosociological table `m_bin` (rows corresponding to taxa and columns corresponding to relevés) this function finds a 2-partition (a partition in two groups) that maximizes TDV, using the Gurobi optimizer.

Gurobi is a commercial software for which a free academic license can be obtained if you are affiliated with a recognized educational institution. Package 'prioritizr' contains a comprehensive vignette (Gurobi Installation Guide), which can guide you trough the process of obtaining a license, installing the Gurobi optimizer, activating the license and eventually installing the R package 'gurobi'.

`optim_tdv_gurobi_k_2()` returns, when the optimization is successful, a 2-partition which is a global maximum of TDV for any 2-partitions of the columns on `m_bin`.

See `tdv()` for an explanation on the Total Differential Value of a phytosociological table.

The function implements two different mixed-integer linear programming formulations of the problem. The formulations differ as one is independent of the size of the obtained groups (t-independent), while the other formulation fixes the size of the obtained groups (t-dependent). The t-dependent formulation is implemented to run Gurobi as many times as necessary to cover all possible group sizes; this approach can result in faster total computation time.

For medium-sized matrices the computation time might become already prohibitive, thus the use of a time limit (`time_limit`) is advisable.

Value

For `formulation = "t-dependent"`, a list with the following components:

status.runs A character vector with Gurobi output status for all the runs.

par A vector with the 2-partition corresponding to the the maximum TDV found by Gurobi.

objval A numeric with the maximum TDV found by Gurobi.

For `formulation = "t-independent"`, a list with the following components:

status A character with Gurobi output status.

par A vector with the 2-partition corresponding to the the maximum TDV found by Gurobi.

objval A numeric with the maximum TDV found by Gurobi.

Author(s)

Jorge Orestes Cerdeira and Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Obtaining the 2-partition that maximizes TDV using the Gurobi solver, by
# mixed-integer linear programming
## Not run:
# Requires the suggested package 'gurobi'
optim_tdv_gurobi_k_2(taxus_bin)

## End(Not run)
```

`optim_tdv_hill_climb` *Total Differential Value optimization using Hill-climbing algorithms*

Description

This function searches for partitions of the columns of a given matrix, optimizing the Total Differential Value (TDV).

Usage

```
optim_tdv_hill_climb(
  m_bin,
  k,
  p_initial = "random",
  n_runs = 1,
  n_sol = 1,
```

```
  maxit = 10,
  min_g_size = 1,
  stoch_first = FALSE,
  stoch_neigh_size = 1,
  stoch_maxit = 100,
  full_output = FALSE,
  verbose = FALSE
)
```

Arguments

m_bin	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
k	A numeric giving the number of desired groups.
p_initial	A vector or a character. A vector of integer numbers with the initial partition of the relevés (i.e., a vector with values from 1 to k, with length equal to the number of columns of m_bin, ascribing each relevé to one of the k groups). By default, p_initial = "random", generates a random initial partition.
n_runs	A numeric giving the number of runs to perform.
n_sol	A numeric giving the number of best solutions to keep in the final output. Defaults to 1.
maxit	A numeric giving the number of iterations of the Hill-climbing optimization.
min_g_size	A numeric. The minimum number of relevés that a group can contain (must be 1 or higher).
stoch_first	A logical. FALSE (the default), performs only Hill-climbing on the 1-neighbours; TRUE first, performs a Stochastic Hill-climbing on n-neighbours (n is defined by the parameter stoch_neigh_size), and only after runs the Hill-climbing search on the 1-neighbours; see description above.
stoch_neigh_size	A numeric giving the size (n) of the n-neighbours for the Stochastic Hill-climbing; only used if stoch_first = TRUE. Defaults to 1.
stoch_maxit	A numeric giving the number of iterations of the Stochastic Hill-climbing optimization; only used if stoch_first = TRUE. Defaults to 100.
full_output	A logical. If FALSE (the default) the best n_sol partitions and respective indices are returned. If TRUE (only available for n_sol = 1) the output will also contain information on the optimization steps (see below).
verbose	A logical. If FALSE nothing is printed during the runs. If TRUE, after each run, the run number is printed as well as and indication if the found partition is a 1-neighbour local maximum.

Details

Given a phytosociological table (m_bin, rows corresponding to taxa and columns corresponding to relevés) this function searches for a k-partition (k defined by the user) optimizing TDV, i.e., searches, using a Hill-climbing algorithm, for patterns of differential taxa by rearranging the relevés into k groups.

The optimization can start from a random partition (`p_ini = "random"`), or from a given partition (`p_ini`, defined by the user or produced by any clustering method, or even a manual classification of the relevés).

In the description given below, a 1-neighbour of a given partition is another partition that can be obtained by simply changing one relevé to a different group. Equivalently a `n`-neighbour of a given partition is another partition obtained ascribing `n` relevés to different groups.

This function implements a Hill-climbing algorithm, where a TDV improvement is searched in each iteration, screening all 1-neighbours, until the given number of maximum iterations (`maxit`) is reached. If `maxit` is not reached but no TDV improvement is possible among all the 1-neighbours of the currently best partition, the search is halted and the current partition is tagged as a local maximum and outputted.

As the screening of all 1-neighbours might be computationally heavy, specially while analysing big tables, optionally, a Stochastic Hill-climbing search can be performed as a first step (`stoch_first = TRUE`). This consists in searching for TDV improvements, by randomly selecting, in each iteration, one `n`-neighbour (`n` defined by the user in the parameter `stoch_neigh_size`), and accepting that `n`-neighbour partition immediately if it improves TDV. This is repeated until a given number of iterations (`stoch_maxit`) is reached. Specially while starting from random partitions, Stochastic Hill-climbing is intended to increase TDV without the computational burden of the full neighbourhood screening, which can be done afterwards, in a second step.

The Hill-climbing or the combination of Stochastic Hill-climbing + Hill-climbing, can be run multiple times by the function (defined in `n_runs`), which consists in a Random-restart Hill-climbing, where `n_sol` best solutions are kept and returned.

As the Hill-climbing algorithm converges easily to local maxima, several runs of the function (i.e., multiple random starts) are advised.

Trimming your table by a 'constancy' range or using the result of other cluster methodologies as input, might help finding interesting partitions. However, after trimming the table by a narrow 'constancy' range, getting a random initial partition with TDV greater than zero might be hard; on such cases using a initial partition from `partition_tdv_grasp()` or `partition_tdv_grdtp()`, or even the result of other clustering strategies, as an input partition might be useful.

Value

If `full_output = FALSE`, a list with (at most) `n_sol` best solutions (equivalent solutions are removed). Each best solution is also a list with the following components:

local_maximum A logical indicating if `par` is a 1-neighbour local maximum.

par A vector with the partition of highest TDV obtained by the Hill-climbing algorithm(s).

tdv A numeric with the TDV of `par`.

If `full_output = TRUE`, a list with just one component (one run only), containing also a list with the following components:

res.stoch A matrix with the iteration number (of the Stochastic Hill-climbing phase), the maximum TDV found until that iteration, and the TDV of the randomly selected n-neighbour in that iteration.

par.stoch A vector with the best partition found in the Stochastic Hill-climbing phase.

tdv.stoch A numeric showing the maximum TDV found in the Stochastic Hill-climbing phase (if selected).

res A matrix with the iteration number (of the Hill-climbing), the maximum TDV found until that iteration, and the highest TDV among all 1-neighbours.

local_maximum A logical indicating if `par` is a 1-neighbour local maximum.

par A vector with the partition of highest TDV obtained by the Hill-climbing algorithm(s).

tdv A numeric with the TDV of `par`.

Author(s)

Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Removing taxa occurring in only one relevé in order to
# reproduce the example in the original article of the data set
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]

# Obtaining a partition that maximizes TDV using the Stochastic Hill-climbing
# and the Hill-climbing algorithms

result <- optim_tdv_hill_climb(
  m_bin = taxus_bin_wmt,
  k = 3,
  n_runs = 7,
  n_sol = 2,
  min_g_size = 3,
  stoch_first = TRUE,
  stoch_maxit = 500,
  verbose = TRUE
)

# Inspect the result. The highest TDV found in the runs.
result[[1]]$tdv
# If result[[1]]$tdv is 0.1958471 you are probably reproducing the three
# groups (Estrela, Gerês and Galicia) from the original article. If not
# try again the optim_tdv_hill_climb function (maybe increasing n_runs).

# Plot the sorted (or tabulated) phytosociological table
tabul1 <- tabulation(
  m_bin = taxus_bin_wmt,
  p = result[[1]]$par,
  taxa_names = rownames(taxus_bin_wmt),
  plot_im = "normal"
)

# Plot the sorted (or tabulated) phytosociological table, also including
# taxa occurring just once in the matrix
```

```
tabul2 <- tabulation(
  m_bin = taxus_bin,
  p = result[[1]]$par,
  taxa_names = rownames(taxus_bin),
  plot_im = "normal"
)
```

`optim_tdv_simul_anne` *Total Differential Value optimization using a Simulated Annealing (and GRASP) algorithm(s)*

Description

This function searches for k-partitions of the columns of a given matrix (i.e., partitions of the columns into k groups), optimizing the Total Differential Value (TDV) using a stochastic global optimization method known as the Simulated Annealing (SANN) algorithm. Optionally, a Greedy Randomized Adaptive Search Procedure (GRASP) can be used to find an initial partition (seed) to be passed to the SANN algorithm.

Usage

```
optim_tdv_simul_anne(
  m_bin,
  k,
  p_initial = NULL,
  n_runs = 10,
  n_sol = 1,
  t_inic = 0.3,
  t_final = 1e-06,
  alpha = 0.05,
  n_iter = 1000,
  use_grasp = TRUE,
  thr = 0.95,
  full_output = FALSE
)
```

Arguments

`m_bin`	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
`k`	A numeric giving the number of desired groups.
`p_initial`	A vector of integer numbers with the partition of the relevés (i.e., a k-partition, consisting in a vector with values from 1 to k, with length equal to the number of columns of `m_bin`, ascribing each relevé to one of the k groups), to be used as initial partition in the Simulated Annealing. For a random partition use `p_initial = "random"`. This argument is ignored if `use_grasp = TRUE`.

n_runs	A numeric giving the number of runs. Defaults to 10.
n_sol	A numeric giving the number of best solutions to keep in the final output (only used if full_output is FALSE; if full_output is TRUE all runs will produce an output). Defaults to 1.
t_inic	A numeric giving the initial temperature. Must be greater than 0 and the maximum admitted value is 1. Defaults to 0.3.
t_final	A numeric giving the final temperature. Must be bounded between 0 and 1. Usually very low values are needed to ensure convergence. Defaults to 0.000001.
alpha	A numeric giving the fraction of temperature drop to be used in the temperature reduction scheme (see Details). Must be bounded between 0 and 1. Defaults to 0.05.
n_iter	A numeric giving the number of iterations. Defaults to 1000.
use_grasp	A logical. Defaults to TRUE. IF TRUE, a GRASP is used to obtain the initial partitions for the Simulated Annealing. If FALSE the user should provide an initial partition or use or use p_initial = "random" for a random one.
thr	A numeric giving a threshold value (from 0 to 1) with the probability used to compute the sample quantile, in order to get the best m_bin columns from which to select one to be include in the GRASP solution (in each step of the procedure). Only needed if use_grasp is TRUE.
full_output	A logical. Defaults to FALSE. If TRUE extra information is presented in the output. See Value.

Details

Given a phytosociological table (m_bin, with rows corresponding to taxa and columns corresponding to relevés) this function searches for a k-partition (k, defined by the user) optimizing the TDV, i.e., searches, using a SANN algorithm (optionally working upon GRASP solutions), for a global maximum of TDV (by rearranging the relevés into k groups).

In the terminology of cluster analysis, taxa correspond to features, variables, or attributes, while relevés correspond to objects or samples.

This function uses two main algorithms:

1. An optional GRASP, which is used to obtain initial solutions (partitions of m_bin) using function partition_tdv_grasp(). Such initial solutions are then submitted to the SANN algorithm.
2. The (main) SANN algorithm, which is used to search for a global maximum of TDV. The initial partition for each run of SANN can be a partition obtained from GRASP (if use_grasp = TRUE) or, (if use_grasp = FALSE), a partition given by the user (using p_initial) or a random partition (using p_initial = "random").

The SANN algorithm decreases the temperature multiplying the current temperature by 1 - alpha according to a predefined schedule, which is automatically calculated from the given values for t_inic, t_final, alpha and n_iter. Specifically, the cooling schedule is obtained calculating the number of times that the temperature has to be decreased in order to approximate t_final starting from t_inic. The number of times that the temperature decreases, say nt, is calculated by the expression:

`floor(log(t_final / t_inic) / log(1 - alpha))`.

Finally, these decreasing stages are scattered through the desired iterations (`n_iter`) homogeneously, by calculating the indices of the iterations that will experience a decrease in temperature using `floor(n_iter / nt * (1:nt))`.

SANN is often seen as an exploratory technique where the temperature settings are challenging and dependent on the problem. This function tries to restrict temperature values taking into account that TDV is always between 0 and 1. Even though, obtaining values of temperature that allow convergence can be challenging. `full_output = TRUE` allows the user to inspect the behaviour of `current.tdv` and check if convergence fails. Generally, convergence failure can be spotted when final SANN TDV values are similar to the initial `current.tdv`, specially when coming from random partitions. In such cases, as a rule of thumb, it is advisable to decrease `t_final` by a factor of 10.

Value

If `full_output = FALSE` (the default), a list with the following components (the GRASP component is only returned if `use_grasp = TRUE`):

GRASP A list with at most `n_sol` components, each one containing also a list with two components:

- **par** A vector with the partition of highest TDV obtained by GRASP;
- **tdv** A numeric with the TDV of `par`.

SANN A list with at most `n_sol` components, each one containing also a list with two components:

- **par** A vector with the partition of highest TDV obtained by the (GRASP +) SANN algorithm(s);
- **tdv** A numeric with the TDV of `par`.

If `full_output = TRUE`, a list with the following components (the GRASP component is only returned if `use_grasp = TRUE`):

GRASP A list with `n_runs` components, each one containing also a list with two components:

- **par** A vector with the partition of highest TDV obtained by GRASP.
- **tdv** A numeric with the TDV of `par`.

SANN A list with `n_runs` components, each one containing also a list with six components:

- **current.tdv** A vector of length `n_iter` with the current TDV of each SANN iteration.
- **alternative.tdv** A vector of length `n_iter` with the alternative TDV used in each SANN iteration.
- **probability** A vector of length `n_iter` with the probability used in each SANN iteration.
- **temperature** A vector of length `n_iter` with the temperature of each SANN iteration.
- **par** A vector with the partition of highest TDV obtained by the (GRASP +) SANN algorithm(s).
- **tdv** A numeric with the TDV of `par`.

Author(s)

Jorge Orestes Cerdeira and Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Removing taxa occurring in only one relevé in order to
# reproduce the example in the original article of the data set
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]

# Obtaining a partition that maximizes TDV using the Simulated Annealing
# algorithm
result <- optim_tdv_simul_anne(
  m_bin = taxus_bin_wmt,
  k = 3,
  p_initial = "random",
  n_runs = 5,
  n_sol = 5,
  use_grasp = FALSE,
  full_output = TRUE
)

# Inspect the result
# The TDV of each run
sapply(result[["SANN"]], function(x) x$tdv)
# The best partition that was found (i.e., with highest TDV)
result[["SANN"]][[1]]$par

# A TDV of 0.1958471 indicates you are probably reproducing the three
# groups (Estrela, Gerês and Galicia) from the original article. A solution
# with TDV = 0.2005789 might also occur, but note that one group has only two
# elements. For now, a minimum group size is not implemented in function
# optim_tdv_simul_anne() as it is in the function optim_tdv_hill_climb().

# Inspect how the optimization progressed (should increase towards the right)
plot(
  result[["SANN"]][[1]]$current.tdv,
  type = "l",
  xlab = "Iteration number",
  ylab = "TDV of the currently accepted solution"
)
for (run in 2:length(result[["SANN"]])) {
  lines(result[["SANN"]][[run]]$current.tdv)
}

# Plot the sorted (or tabulated) phytosociological table, using the best
# partition that was found
tabul <- tabulation(
  m_bin = taxus_bin_wmt,
  p = result[["SANN"]][[1]]$par,
  taxa_names = rownames(taxus_bin_wmt),
  plot_im = "normal"
)
```

`partition_tdv_grasp` *Obtain a partition using a GRASP algorithm*

Description

This function obtains a partition of the columns of a given phytosociological matrix, aiming at high values of the Total Differential Value (TDV) using a GRASP algorithm.

Usage

```
partition_tdv_grasp(m_bin, k, thr = 0.95, verify = TRUE)
```

Arguments

`m_bin`	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
`k`	A numeric giving the number of desired groups.
`thr`	A numeric giving a threshold value (from 0 to 1) with the probability used to compute the sample quantile, in order to get the best `m_bin` columns from which to select one to be include in the GRASP solution (in each step of the procedure).
`verify`	A logical. If `TRUE` (the default) the function verifies if basic features of `m_bin` data structure are met. Otherwise if `FALSE`.

Details

This function uses a Greedy Randomized Adaptive Search Procedure (GRASP) to obtain a partition of `m_bin`. Given a phytosociological table (`m_bin`, with rows corresponding to taxa and columns corresponding to relevés) this function searches for a `k`-partition (`k`, defined by the user) aiming at high values of the TDV. See `tdv()` for an explanation on the TDV of a phytosociological table.

With `thr = 1`, the algorithm corresponds to the Greedy algorithm.

Value

A numeric vector, which length is the same as the number of columns of `m_bin`, with numbers from 1 to `k`, representing the group to which the respective column was ascribed.

Author(s)

Jorge Orestes Cerdeira and Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Obtaining a partition based on the GRASP algorithm
partition_tdv_grasp(taxus_bin, 3)
```

`partition_tdv_grdtp` *Obtain a partition using a Greedy-type algorithm*

Description

This function obtains a partition of the columns of a given phytosociological matrix, aiming at high values of the Total Differential Value (TDV), implementing a Greedy-type algorithm.

Usage

```
partition_tdv_grdtp(m_bin, k, verify = TRUE)
```

Arguments

`m_bin`	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
`k`	A numeric giving the number of desired groups.
`verify`	A logical. If `TRUE` (the default) the function verifies if basic features of `m_bin` data structure are met. Otherwise if `FALSE`.

Details

Given the phytosociological table `m_bin` (rows corresponding to taxa and columns corresponding to relevés), this function uses a Greedy-type algorithm (a simplified version of the Greedy algorithm) to obtain a `k`-partition (`k`, defined by the user) of the columns of `m_bin`, aiming at high values of TDV. The algorithm operates in the following way: Firstly, `k` columns are selected randomly to work as seeds for each one of the desired `k` groups. Secondly, one of the remaining columns is selected randomly and added to the partition group which maximizes the upcoming TDV. This second step is repeated until all columns are placed in a group of the `k`-partition.

This function is expected to perform faster than `partition_tdv_grasp()`, yet returning worse partitions in terms of TDV. For the (true) Greedy algorithm see `partition_tdv_grasp()`. See `tdv()` for an explanation on the TDV of a phytosociological table.

Value

A numeric vector, which length is the same as the number of columns of `m_bin`, with numbers from 1 to `k`, representing the group to which the respective column was ascribed.

Author(s)

Jorge Orestes Cerdeira and Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Obtaining a partiton based on a Greedy-type algorithm
partition_tdv_grdtp(taxus_bin, 3)
```

`tabulation` *Rearrange a phytosociological table, showing differential taxa on top*

Description

This function reorders a phytosociological table rows using, firstly, the increasing number of groups in which a taxon occurs, and secondly, the decreasing sum of the inner frequency of presences of each taxon (see `tdv()`). The columns are also reordered, simply using the increasing number of the respective group membership.

Usage

```
tabulation(
  m_bin,
  p,
  taxa_names,
  plot_im = NULL,
  palette = "Vik",
  greyout = TRUE,
  greyout_colour = "grey"
)
```

Arguments

`m_bin`	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
`p`	A vector of integer numbers with the partition of the relevés (i.e., a k-partition, consisting in a vector with values from 1 to k, with length equal to the number of columns of `m_bin`, ascribing each relevé to one of the k groups).
`taxa_names`	A character vector (with length equal to the number of rows of `m_bin`) with the taxa names.
`plot_im`	By default, `NULL`, returns without plotting. If `plot_im = "normal"`, plots an image of the tabulated matrix. If `plot_im = "condensed"`, plots an image of the tabulated matrix but presenting sets of differential taxa as solid coloured blocks.
`palette`	A character with the name of the colour palette (one of `grDevices::hcl.pals()` to be passed to `grDevices::hcl.colors()`. Defaults to "Vik".

greyout	A logical. If TRUE (the default), non-differential taxa are greyed out (using the colour defined by greyout_colour). If FALSE, non-differential taxa is depicted with the respective group colours.
greyout_colour	A character with the name of the colour to use for non-differential taxa. Defaults to "grey".

Details

The function accepts a phytosociological table (m_bin), a k-partition of its columns (p) and the names of the taxa (corresponding to the rows of m_bin), returning a rearranged/reordered matrix (and plotting optionally).

Value

If plot_im = NULL, a list with the following components:

taxa.names The given taxa_names

taxa.ord A vector with the order of the rows/taxa.

tabulated The rearranged/reordered m_bin matrix.

condensed The matrix used to create the "condensed" image.

If plot_im = "normal", it returns the above list and, additionally, plots an image of the tabulated matrix. If plot_im = "condensed", it returns the above list and, additionally, plots an image of the tabulated matrix, but presenting the sets of differential taxa as solid coloured blocks of equal width.

Author(s)

Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Creating a group partition, as presented in the original article of the
# data set
groups <- rep(c(1, 2, 3), c(3, 11, 19))

# Removing taxa occurring in only one relevé in order to
# reproduce exactly the example in the original article of the data set
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]

# Sorting the phytosociological table, putting exclusive taxa in the top and
# plotting an image of it
tabul <- tabulation(
  m_bin = taxus_bin_wmt,
  p = groups,
  taxa_names = rownames(taxus_bin_wmt),
  plot_im = "normal",
  palette = "Zissou 1"
```

```
)

# Inspect the first rows and columns of the reordered phytosociological table
head(tabul$tabulated, n = c(5, 5))
```

`taxus_bin` Taxus baccata *forests*

Description

A binary phytosociological table containing relevés of *Taxus baccata* forests, from the northwest of the Iberian Peninsula.

Usage

```
taxus_bin
```

Format

A matrix with 209 rows and 33 columns. Each column corresponds to a phytosociological relevé and each row corresponds to a taxon. Values in the matrix denote presences (1) and absences (0).

Source

Portela-Pereira E., Monteiro-Henriques T., Casas C., Forner N., Garcia-Cabral I., Fonseca J.P. & Neto C. 2021. *Teixedos no noroeste da Península Ibérica*. Finisterra 56(117): 127-150. doi:10.18055/FINIS18102.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Inspect the first rows and columns of taxus_bin
head(taxus_bin, n = c(5, 5))
```

tdv *The Total Differential Value of a phytosociological table*

Description

Given a phytosociological table and a partition of its columns, this function calculates the respective Total Differential Value (TDV).

Usage

```
tdv(m_bin, p, output_type = "normal")
```

Arguments

m_bin	A matrix. A phytosociological table of 0s (absences) and 1s (presences), where rows correspond to taxa and columns correspond to relevés.
p	A vector of integer numbers with the partition of the relevés (i.e., a k-partition, consisting in a vector with values from 1 to k, with length equal to the number of columns of m_bin, ascribing each relevé to one of the k groups).
output_type	A character determining the amount of information returned by the function and also the amount of pre-validations. Possible values are "normal" (the default), "fast" and "full".

Details

The function accepts a phytosociological table (m_bin) and a k-partition of its columns (p), returning the corresponding TDV (Monteiro -Henriques 2025).

TDV was proposed by Monteiro-Henriques and Bellu (2014). Monteiro-Henriques (2016) proposed TDV1, modifying TDV slightly with the objective of ensuring a value from 0 to 1. Yet, TDV is always within that range. In practice, both TDV and TDV1 have 0 as possible minimum value and 1 as possible maximum value, but TDV1 reduces further the contribution of differential taxa present in more than one group. TDV is then implemented here, for parsimony.

TDV is calculated using the $DiffVal$ index for each (and all) of the taxa present in a tabulated phytosociological table M (also called sorted table). $DiffVal$ index aims at characterizing how well a taxon works as a differential taxon in a such tabulated phytosociological table (for more information on differential taxa see Mueller-Dombois & Ellenberg, 1974).

An archetypal differential taxon of a certain group g of the partition p (a partition on the columns of M) is the one present in all relevés of group g, and absent from all the other groups of that partition. Therefore, $DiffVal$ has two components, an inner one ($\frac{a}{b}$), which measures the presence of the taxon inside each of the groups, and an outer one ($\frac{c}{d}$), which measures the relevant absences of the taxon outside of each of the groups. Specifically, given a partition p with k groups, $DiffVal$ is calculated for each taxon s as:

$$DiffVal_{s,p} = \frac{1}{e} \sum_{g=1}^{k} \frac{a}{b} \frac{c}{d}$$

where:

- a, is the total number of presences of taxon s within group g.
- b, is the total number of relevés of group g.
- c, is the total number of differentiating absences of taxon s, i.e., absences coming from the groups other than g from which the taxon s is completely absent.
- d, is the total number of relevés of all groups but g (i.e., the total number of relevés in the table - b).
- e, is the total number of groups in which the taxon s occurs at least once.

Therefore, for each taxon s and for each group g, the $DiffVal$ index evaluates:

- $\frac{a}{b}$, i.e., the frequency of the presences of taxon s, relative to the size of group g; commonly called 'relative frequency.' $\frac{a}{b}$ is only 1 if and only if taxon s occurs in all the relevés of group g.
- $\frac{c}{d}$, i.e., the frequency of the differentiating absences of taxon s outside group g, relative to the sum of sizes of all groups but g. *Nota bene*: absences in c are counted outside the group g but only in the groups from which taxon s is completely absent (these are the relevant absences, which produce differentiation among groups); in practice c corresponds to the sum of the sizes of all groups other than g that are empty. $\frac{c}{d}$ is 1 if and only if the taxon s is absent from all groups but g.

Finally, $\frac{1}{e}$ ensures that $DiffVal$ is a value from 0 to 1.

The Total Differential Value (TDV or $TotDiffVal$) of a phytosociological table M tabulated/sorted by the partition p is:

$$TDV_{M,p} = \frac{1}{n} \sum_{i=1}^{n} Diffval_{i,p}$$

where:

- n, is the number of taxa in table M.

The division by the number of taxa present in M ensures that TDV remains in the [0,1] interval (as $DiffVal$ is also in the same interval).

Value

If `output_type = "normal"` (the default) pre-validations are done and a list is returned, with the following components:

ifp A matrix with the $\frac{a}{b}$ values for each taxon in each group, for short called the 'inner frequency of presences'.

ofda A matrix with the $\frac{c}{d}$ values for each taxon in each group, for short called the 'outer frequency of differentiating absences'.

e A vector with the e values for each taxon, i.e., the number of groups containing that taxon.

diffval A matrix with the $DiffVal$ for each taxon.

tdv A numeric with the TDV of matrix `m_bin`, given the partition `p`.

If `output_type = "full"`, some extra components are added to the output: `afg`, `empty.size`, `gct` (= e) and `i.mul`. These are intermediate matrices used in the computation of TDV.

If `output_type = "fast"`, only TDV is returned and no pre-validations are done.

Author(s)

Tiago Monteiro-Henriques. E-mail: <tmh.dev@icloud.com>.

References

Monteiro-Henriques T, Bellu A. 2014. *An optimization approach to the production of differentiated tables based on new differentiability measures*. 23rd EVS European Vegetation Survey. Presented orally. Ljubljana, Slovenia.

Monteiro-Henriques T 2016. *A bunch of R functions to assist phytosociological tabulation*. 25th Meeting of European Vegetation Survey. Presented in poster. Rome. Italy.

Monteiro-Henriques T 2025. *TDV-optimization: A novel numerical method for phytosociological tabulation*. Vegetation Classification and Survey 6: 99-127. DOI: doi:10.3897/VCS.140466

Mueller-Dombois D, Ellenberg H 1974. *Aims and Methods of Vegetation Ecology*. New York: John Wiley & Sons.

Examples

```
# Getting the Taxus baccata forests data set
data(taxus_bin)

# Creating a group partition, as the one presented in the original article of
# the data set
groups <- rep(c(1, 2, 3), c(3, 11, 19))

# Removing taxa occurring in only one relevé, in order to reproduce exactly
# the example in the original article of the data set
taxus_bin_wmt <- taxus_bin[rowSums(taxus_bin) > 1, ]

# Calculating TDV
result <- tdv(taxus_bin_wmt, groups)

# This is the TDV
result$tdv
# This is TDV1, reproducing exactly the value from the original article
sum(result$diffval / result$e) / nrow(taxus_bin_wmt)
```

Index

∗ **datasets**
taxus_bin, 21

bigdata_tdv, 2

explore_tabulation, 4

grDevices::hcl.colors(), *4*, *19*
grDevices::hcl.pals(), *4*, *19*

identical_partition, 5
internal_assignment, 6

optim_tdv_gurobi_k_2, 8
optim_tdv_gurobi_k_2(), *8*
optim_tdv_hill_climb, 9
optim_tdv_simul_anne, 13

parallel::mclapply(), *3*
partition_tdv_grasp, 17
partition_tdv_grasp(), *11*, *14*, *18*
partition_tdv_grdtp, 18
partition_tdv_grdtp(), *11*

tabulation, 19
tabulation(), *4, 5*
taxus_bin, 21
tdv, 22
tdv(), *3*, *8*, *17–19*