Package: wompwomp (via r-universe)

October 21, 2026

Version 1.0.0

Date 2026-09-06

Title Optimizing Alluvial Plots

Description Sort k-partite graphs with node order, layer order, and node grouping optimized with a heuristic to (nearly) minimize edge crossings, and render the result as an alluvial plot. Useful for improving visualizations with alluvial plots by ``untangling'' the graphs.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Imports dplyr, igraph, tibble, rlang, tidyr, TSP, stats, tidyselect, vctrs, utils, Rcpp, ggplot2, ggalluvial, ggforce, magrittr

LinkingTo Rcpp

Depends R (>= 4.0.0)

Suggests testthat, vdiffr, magick, knitr, rmarkdown, gtools, mclust, stringr, ggfittext, ggrastr, sessioninfo

Config/testthat/edition 3

URL https://github.com/pachterlab/wompwomp, https://cran.r-project.org/package=wompwomp

BugReports https://github.com/pachterlab/wompwomp/issues

Config/roxygen2/version 8.1.0

RoxygenNote 7.3.3

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev libglpk-dev libicu-dev libxml2-dev

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

Date/Publication 2026-09-21 18:06:24 UTC

RemoteUrl https://github.com/pachterlab/wompwomp

RemoteRef HEAD

RemoteSha 88fba6059ffa6c5a5e0f2ccb3ed97bcf69e562b9

Contents

compute_color_agreement 2
compute_crossing_objective 3
get_lode_clusters 5
get_lode_clusters_options 6
lode_cluster_pal 7
plot_alluvial 8
plot_alluvial-internals 14
prep_for_lodes 14
sort_to_uncross 16
sort_to_uncross_options 19

Index **22**

`compute_color_agreement`

Color agreement of a stratum coloring

Description

Computes the W_LOMP color-agreement objective $\mathcal{M}$ for a coloring produced by `get_lode_clusters()`: the total weight of the observations whose blocks in two different layers were given the same color.

Usage

```
compute_color_agreement(
  data,
  cols,
  mapping,
  wt = NULL,
  adjacent_only = FALSE,
  missing_value = "Missing"
)
```

Arguments

`data`	A data frame or tibble. Must be in one of two formats: (1) wt == NULL: each row represents an entity and each column a grouping; or (2) wt != NULL: each row represents a combination of groupings and the column `wt` holds the number of entities in that combination.
`cols`	Character vector. Column names from `data` used in graphing.
`mapping`	List. Output of `get_lode_clusters()`: a list indexed by column name, each entry a list mapping that column's stratum values to color ids.
`wt`	Optional character. Column name holding the weight of each row.

`adjacent_only` Logical. If TRUE, sum only over adjacent layer pairs `(i, i+1)` in the order given by `cols`, rather than all pairs.

`missing_value` Character. Name of the stratum of missing (`NA`) entries in `mapping`; pass the value given to `get_lode_clusters_options()` (default `"Missing"`).

Details

Writing $W_{bb'}^{(i,j)}$ for the co-occurrence weight of block b of layer i and block b' of layer j, and $c^{(i)}$ for the color assigned to the blocks of layer i,

$$\mathcal{M} = \sum_{i<j} \sum_{b,b'} \delta\Big(c^{(i)}(b), c^{(j)}(b')\Big) \cdot W_{bb'}^{(i,j)}.$$

The sum runs over every unordered pair of layers, matching the graph that `get_lode_clusters()` clusters (which also uses every pair, not only adjacent ones). Set `adjacent_only = TRUE` to restrict it to the $m - 1$ adjacent pairs, i.e. to the layer pairs that are actually drawn side by side.

$\mathcal{M}$ is a diagnostic, not something the coloring maximizes: because colors may repeat within a layer, $\mathcal{M}$ is maximized trivially by painting every block the same color. It is reported so a coloring can be compared against another coloring of the same data.

Value

A list with:

`color_agreement` The scalar $\mathcal{M}$.

`per_pair` A data frame with one row per layer pair, giving the contribution $\mathcal{M}^{(i,j)}$ of that pair and the total co-occurrence weight available to it.

Examples

```
set.seed(429144)
data <- data.frame(
  method1 = factor(LETTERS[sample(1:3, 100, TRUE)]),
  method2 = factor(LETTERS[27 - sample(1:3, 100, TRUE)])
)
mapping <- get_lode_clusters(data, cols = c("method1", "method2"))
compute_color_agreement(data, cols = c("method1", "method2"), mapping = mapping)
```

`compute_crossing_objective` *Determine overlapping edges*

Description

Determine overlapping edges of k-partite graph.

Usage

```
compute_crossing_objective(
  data,
  cols = NULL,
  wt = "value",
  weighted_metric = TRUE,
  verbose = FALSE
)
```

Arguments

data	A data frame, tibble, or CSV file path. Must be in one of two formats: (1) wt == NULL: Each row represents an entity, each column represents a grouping, and each entry represents the membership of the entity in that row to the grouping in that column. Must contain at least two columns (two cols). (2) wt != NULL: Each row represents a combination of groupings, each column from cols represents a grouping, and the column wt represents the number of entities in that combination of groupings. Must contain at least three columns (two cols, one wt).
cols	Optional character vector. Vector of column names from data to be used in graphing (i.e., alluvial plotting). Mutually exclusive with column1 and column2.
wt	Optional character. Column name from data that contains the weights of each combination of groupings if data is in format (2) (see above). If null, then sets weighted_metric to FALSE.
weighted_metric	Logical. Determines if the objective is total number of edge crossings (weighted_metric=FALSE) or sum of product of overlapping edge weights (weighted_metric=TRUE).
verbose	Logical. If TRUE, will display messages during the function.

Value

If return_weighted_layer_free_objective is FALSE (default): A list of values, as follows: 'lode_df': A data frame containing the following columns:

- alluvium: A specific alluvium/edge.
- count: The weight of the alluvium/edge.
- x1, x2, ...: Each xi represents the x position of axis/layer i.
- y1, y2, ...: Each yi represents the height of a lode in axis/layer i.
- stratum1, stratum2, ...: Each stratumi represents the stratum through which the alluvial crosses in axis/layer i.
- weight1: The weight of the first alluvium, corresponding to the 'count' column in lode_df.
- weight2: The weight of the second alluvium, corresponding to the 'count' column in lode_df. 'output_objective': An integer representing the sum of products of overlapping edge weights.

Examples

```
data <- data.frame(method1 = sample(1:3, 100, TRUE), method2 = sample(1:3, 100, TRUE))
data <- sort_to_uncross(data, cols = c("method1", "method2"), method = "tsp")
result <- compute_crossing_objective(data, cols = c("method1", "method2"))
```

`get_lode_clusters` *Color alluvia*

Description

Colors a dataframe with the algorithm specified by `method`.

Usage

```
get_lode_clusters(
  data,
  cols,
  wt = NULL,
  method = "advanced",
  resolution = 1,
  verbose = FALSE,
  options = NULL
)
```

Arguments

`data`	A data frame, tibble, or CSV file path. Must be in one of two formats: (1) wt == NULL: Each row represents an entity, each column represents a grouping, and each entry represents the membership of the entity in that row to the grouping in that column. Must contain at least two columns (two cols). (2) wt != NULL: Each row represents a combination of groupings, each column from `cols` represents a grouping, and the column `wt` represents the number of entities in that combination of groupings. Must contain at least three columns (two `cols`, one `wt`).
`cols`	Character vector. Vector of column names from `data` to be used in graphing (i.e., alluvial plotting).
`wt`	Optional character. Column name from `data` that contains the weights of each combination of groupings if `data` is in format (2) (see above).
`method`	Character. Matching colors methods. Choices are 'advanced' (default), 'none', 'left', 'right', or any value in `cols`.
`resolution`	Numeric If `method == 'advanced'`, then choose resolution for the graph clustering algorithm. Higher resolutions correspond to more distinct colors.
`verbose`	Logical. If TRUE, will display messages during the function.
`options`	Additional arguments. See get_lode_clusters_options

Value

A nested list mapping each stratum to a color. The outer list is indexed by axis names, and the inner lists are indexed by stratum names. Each color is represented by an integer that is mapped to a color by the `mapping` parameter in `scale_fill_manual()`.

Examples

```
set.seed(429144)
data <- data.frame(
  method1 = factor(LETTERS[sample(1:3, 100, TRUE)]),
  method2 = factor(LETTERS[27 - sample(1:3, 100, TRUE)])
)
lapply(data, levels)
cluster_mapping <- data |>
  get_lode_clusters(cols = c(method1, method2))
```

---

get_lode_clusters_options

Control Options for `get_lode_clusters()`

Description

Creates a list of control parameters that modify the behavior of `get_lode_clusters()`. These options allow fine-grained control over the coloring algorithm without cluttering the main function interface.

Usage

```
get_lode_clusters_options(
  method_advanced_option = c("leiden", "louvain"),
  cutoff = 0.5,
  preprocess_data = TRUE,
  missing_value = "Missing",
  print_params = FALSE
)
```

Arguments

`method_advanced_option`
: Character. When `method == "advanced"`, selects the clustering algorithm. Options: `"leiden"` (default) or `"louvain"`.

`cutoff`
: Numeric. If using a non-advanced algorithm, similarity score threshold for deciding whether to reuse a color or assign a new one.

`preprocess_data`
: Logical. If TRUE (default), preprocess data with `prep_for_lodes()`.

missing_value	Character. Missing (NA) entries of cols, including an explicit NA factor level such as sort_to_uncross() returns with missing_value = NA, are colored as a stratum of their own, and this is the name under which that stratum appears in the mapping (default "Missing"). If a column that has missing entries already uses the name as a value, the two are merged into one stratum, with a warning; choose a name the data do not use to keep them apart. Only applies when preprocess_data is TRUE.
print_params	Logical. If TRUE, prints parameters during execution.

Value

A named list of control parameters for use in get_lode_clusters().

Examples

```
data <- data.frame(method1 = sample(1:3, 100, TRUE), method2 = sample(1:3, 100, TRUE))
opts <- get_lode_clusters_options(cutoff = 0.3, method_advanced_option = "louvain")
mapping <- get_lode_clusters(data = data, cols = c('method1', 'method2'), options = opts)
```

---

lode_cluster_pal *Make stratum color list*

Description

Convert color mapping made by get_lode_clusters into list for scale_fill_manual

Usage

```
lode_cluster_pal(data, cols, mapping, color_palette = NULL, per_axis = FALSE)
```

Arguments

data	A data frame with columns corresponding to axes, with factors indicating sorting. (eg run through sort_to_uncross)
cols	Character vector. Vector of column names from data to be used in graphing (i.e., alluvial plotting).
mapping	List. Output from get_lode_clusters.
color_palette	Optional named list or vector mapping values in the graphing columns to colors. Overrides default palette.
per_axis	Logical. If FALSE (default), returns one color per distinct stratum value across all columns (a flat named vector). If TRUE, returns a named list with one entry per column, each a named vector of colors for that column's levels – so a value appearing in two columns can take a different color in each (as when a method over-splits a group).

Value

A named vector of colors, or (if `per_axis`) a list of them. An explicit `NA` factor level (as `sort_to_uncross()` returns with `missing_value = NA`) has no name and gets no entry; ggplot2 scales color it with their `na.value`.

Examples

```
# Example 1
data <- data.frame(method1 = sample(1:3, 100, TRUE), method2 = sample(1:3, 100, TRUE))
cols = c("method1", "method2")
clus_df_gather <- sort_to_uncross(data, cols = cols, method = "tsp")
color_mapping <- get_lode_clusters(data = clus_df_gather, cols = cols)
color_list <- lode_cluster_pal(data = clus_df_gather, cols = cols, mapping = color_mapping)
```

---

plot_alluvial *Generate an Alluvial Plot with Minimal Cluster Cross-over*

Description

Creates a two-axis alluvial plot to visualize the relationship between two categorical groupings (e.g., cluster assignments from different methods), minimizing crossover and aligning clusters based on agreement.

Usage

```
plot_alluvial(
  df,
  graphing_columns = NULL,
  column1 = NULL,
  column2 = NULL,
  column_weights = NULL,
  sorting_algorithm = "neighbornet",
  optimize_column_order = TRUE,
  optimize_column_order_per_cycle = FALSE,
  alpha = 2,
  beta = alpha,
  weight_scalar = 5e+05,
  matrix_initialization_value = NULL,
  same_side_matrix_initialization_value = NULL,
  matrix_initialization_value_column_order = 1e+06,
  weight_scalar_column_order = 1,
  column_sorting_metric = "edge_crossing",
  column_sorting_algorithm = "neighbornet",
  weighted = TRUE,
  cycle_start_positions = NULL,
  fixed_column = NULL,
```

```
  random_initializations = 1,
  color_boxes = TRUE,
  color_bands = FALSE,
  color_list = NULL,
  color_band_list = NULL,
  color_band_column = NULL,
  color_val = NULL,
  color_band_boundary = FALSE,
  coloring_algorithm = "advanced",
  coloring_algorithm_advanced_option = "leiden",
  resolution = 1,
  cutoff = 0.5,
  alluvial_alpha = 0.5,
  include_labels_in_boxes = TRUE,
  include_axis_titles = TRUE,
  include_group_sizes = FALSE,
  output_plot_path = NULL,
  output_df_path = NULL,
  preprocess_data = TRUE,
  default_sorting = "alphabetical",
  missing_value = "Missing",
  box_width = 1/3,
  text_width = 1/4,
  min_text = 4,
  text_size = 14,
  auto_adjust_text = TRUE,
  axis_text_size = 2,
  axis_text_vjust = 0,
  save_height = 6,
  save_width = 6,
  dpi = 300,
  rasterise_alluvia = FALSE,
  keep_y_labels = FALSE,
  keep_x_labels = TRUE,
  box_line_width = 1,
  verbose = FALSE,
  print_params = FALSE,
  add_legend = FALSE,
  legend_loc = "right",
  flip_xy = FALSE
)
```

Arguments

df	A data frame, tibble, or CSV file path. Must be in one of two formats: (1) column_weights == NULL: Each row represents an entity, each column represents a grouping, and each entry represents the membership of the entity in that row to the grouping in that column. Must contain at least two columns (two graphing_columns). (2) column_weights != NULL: Each row represents

a combination of groupings, each column from graphing_columns represents a grouping, and the column column_weights represents the number of entities in that combination of groupings. Must contain at least three columns (two graphing_columns, one column_weights).

graphing_columns
: Optional character vector. Vector of column names from df to be used in graphing (i.e., alluvial plotting). Mutually exclusive with column1 and column2.

column1
: Optional character. Can be used along with column2 in place of graphing_columns if working with two columns only. Mutually exclusive with graphing_columns.

column2
: Optional character. Can be used along with column1 in place of graphing_columns if working with two columns only. Mutually exclusive with graphing_columns.

column_weights
: Optional character. Column name from df that contains the weights of each combination of groupings if df is in format (2) (see above).

sorting_algorithm
: Character. Algorithm with which to sort the values in the dataframe (default 'neighbornet'). Can choose from: 'neighbornet', 'tsp', 'greedy', 'barycenter', 'median', 'random', 'none'. 'neighbornet' performs sorting with Neighbor-Net (Bryant and Moulton, 2004). 'tsp' performs Traveling Salesman Problem solver from the TSP package. 'greedy', 'barycenter' and 'median' sweep across the axes, reordering each against its neighbor: 'greedy' places each block as close as possible to its heaviest neighbor, 'barycenter'/'median' at the weighted mean/median position of its neighbors. All methods accept any number of graphing_columns. 'random' randomly maps blocks. 'none' keeps the mappings as-is when passed into the function. See sort_to_uncross() for details and for the deprecated names 'greedy_wolf', 'greedy_wblf', 'barycenter_one_sided' and 'median_one_sided'.

optimize_column_order
: Logical. If TRUE, will optimize the order of graphing_columns to minimize edge overlap. Only applies when length(graphing_columns) > 2.

optimize_column_order_per_cycle
: Logical. If TRUE, will optimize the order of graphing_columns to minimize edge overlap upon each cycle. If FALSE, will optimize the order of graphing_columns to minimize edge overlap on the beginning cycle only. Only applies when sorting_algorithm == 'neighbornet' or 'tsp' and length(graphing_columns) > 2.

alpha
: Positive number (default 2). Ratio between the distance assigned to two blocks in different axes that share no observations and the scale of the $-\log(\text{edge weight})$ distances between blocks that do. Together with beta, this is the tuning knob of the block distance matrix; see sort_to_uncross(). Only applies when sorting_algorithm == 'neighbornet' or 'tsp'.

beta
: Positive number (default alpha). Ratio between the distance assigned to two distinct blocks of the same axis and the scale of the edge-weight distances. See sort_to_uncross(). Only applies when sorting_algorithm == 'neighbornet' or 'tsp'.

weight_scalar
: Advanced. Positive number setting the absolute scale of the block distance matrix (the constant c multiplying $-\log(\text{edge weight})$). The cycle depends only

on the ratios alpha and beta, so this rarely needs changing. Only applies when sorting_algorithm == 'neighbornet' or 'tsp'.

matrix_initialization_value
: Advanced. Distance between blocks in different axes that share no observations. NULL (default) derives it as alpha * weight_scalar; a value overrides alpha. Only applies when sorting_algorithm == 'neighbornet' or 'tsp'.

same_side_matrix_initialization_value
: Advanced. Distance between distinct blocks of the same axis. NULL (default) derives it as beta * weight_scalar; a value overrides beta. Only applies when sorting_algorithm == 'neighbornet' or 'tsp'.

matrix_initialization_value_column_order
: Positive integer. Initialized value in distance matrix for optimizing column order. Only applies when sorting_algorithm == 'neighbornet' or 'tsp' and optimize_column_order is TRUE.

weight_scalar_column_order
: Positive integer. Scalar with which to loss function after taking their log1p in the distance matrix for optimizing column order. Only applies when sorting_algorithm == 'neighbornet' or 'tsp' and optimize_column_order is TRUE.

column_sorting_metric
: Character. Metric to use for determining column order. Options are "edge_crossing" (default) or "ARI". Only applies when sorting_algorithm == 'neighbornet' or 'tsp' and optimize_column_order is TRUE.

column_sorting_algorithm
: Character. Algorithm to use for determining column order. Options are "neighbornet" (default) or "tsp". Only applies when sorting_algorithm == 'neighbornet' or 'tsp' and optimize_column_order is TRUE.

weighted
: Logical. Weighted objective

cycle_start_positions
: Set. Cycle start positions to consider. Anything outside this set will be skipped. Only applies when sorting_algorithm == 'neighbornet' or 'tsp'.

fixed_column
: Optional character or integer vector. Names, or positions in graphing_columns, of the axes whose stratum order is kept as given while the other axes are sorted. NULL (default) sorts every axis. See sort_to_uncross().

random_initializations
: Integer. Number of initializations of the stratum positions of the non-fixed axes (the first from the incoming order, the rest random), keeping the one with the fewest crossings. Only applies when sorting_algorithm %in% c('greedy', 'barycenter', 'median').

color_boxes
: Logical. Whether to color the strata/boxes (representing groups).

color_bands
: Logical. Whether to color the alluvia/edges (connecting the strata).

color_list
: Optional named list or vector of colors to override default group colors.

color_band_list
: Optional named list or vector of colors to override default band colors.

color_band_column
: Optional Character. Which column to use for coloring bands.

`color_val` Optional named list where the entries are colors and the names correspond to values of the dataframe that should use those colors

`color_band_boundary`
Logical. Whether or not to color boundaries between bands

`coloring_algorithm`
Character. Matching colors methods. Choices are 'advanced' (default), 'none', 'left', 'right', or any value in `graphing_columns`.

`coloring_algorithm_advanced_option`
Character. If `coloring_algorithm == 'advanced'`, then choose graph clustering algorithm. Choices are 'leiden' (default) or 'louvain'.

`resolution` Numeric If `coloring_algorithm == 'advanced'`, then choose resolution for the graph clustering algorithm. Affects coloring of both bands and boxes.

`cutoff` Numeric If `coloring_algorithm != 'none'` and `coloring_algorithm != 'advanced'`, sets the cutoff for color matching, below which a new color will be assigned.

`alluvial_alpha` Numeric between 0 and 1. Transparency level for the alluvial bands.

`include_labels_in_boxes`
Logical. Whether to include text labels inside the rectangular group boxes.

`include_axis_titles`
Logical. Whether to display axis titles for column1 and column2.

`include_group_sizes`
Logical. If `TRUE`, includes group sizes in the labels (e.g., "Group A (42)").

`output_plot_path`
Character. File path to save the plot (e.g., "plot.png"). If `NULL`, then will not be saved.

`output_df_path` Optional character. Output path for the output data frame, in CSV format. If `NULL`, then will not be saved.

`preprocess_data`
Logical. If TRUE, will preprocess the data with the `prep_for_lodes` function.

`default_sorting`
Character. Default column sorting in prep_for_lodes if integer columns do not exist. Options are 'alphabetical' (default), 'reverse_alphabetical', 'increasing', 'decreasing', 'random'.

`missing_value` Character. Missing (`NA`) entries of the graphing columns are drawn as a stratum of their own, and this is its label (default `"Missing"`). If a column that has missing entries already uses the label as a value, the two are merged into one stratum, with a warning; choose a label the data do not use to keep them apart. Only applies when `preprocess_data` is TRUE.

`box_width` Numeric between 0 and 1. Box width

`text_width` Numeric between 0 and 1. Text width

`min_text` Integer greater than 0. Min text

`text_size` Integer greater than 0. Text size (works whether auto_adjust_text is TRUE or FALSE).

`auto_adjust_text`
Logical. Whether to automatically adjust text size to fit in box.

axis_text_size	Integer greater than 0. Axis text size
axis_text_vjust	Integer. Axis text vjust
save_height	Integer greater than 0. Save height, in inches
save_width	Integer greater than 0. Save width, in inches
dpi	Integer greater than 0. DPI for output_plot_path, if output_plot_path is a raster image or rasterise_alluvia is TRUE
rasterise_alluvia	Logical. Whether to rasterize the alluvia if output_plot_path is a PDF. Can save space if DPI low enough
keep_y_labels	Keep y labels
keep_x_labels	Keep x labels
box_line_width	Box line width
verbose	Logical. If TRUE, will display messages during the function.
print_params	Logical. If TRUE, will print function params.
add_legend	Logical. If TRUE, will generate a legend of the colors of boxes and alluvial
legend_loc	Character. Location of legend. Only applies if add_legened == TRUE. Choices are 'right' (default), 'left', 'bottom', 'top'
flip_xy	Logical. Flip x and y (rotate plot 90 degrees).

Value

A ggplot2 object representing the alluvial plot.

Examples

```
# Example 1: df format 1
df <- data.frame(method1 = sample(1:3, 100, TRUE), method2 = sample(1:3, 100, TRUE))
p <- plot_alluvial(df,
    graphing_columns = c("method1", "method2"),
    sorting_algorithm = "tsp",
    column_sorting_algorithm = "tsp",
    coloring_algorithm = "right"
)

# Example 2: df format 2
df <- data.frame(method1 = sample(1:3, 100, TRUE), method2 = sample(1:3, 100, TRUE))
clus_df_gather <- df |>
   dplyr::mutate_if(is.numeric, function(x) factor(x, levels = as.character(sort(unique(x))))) |>
    dplyr::group_by_all() |>
    dplyr::count(name = "value")
p <- plot_alluvial(
    clus_df_gather,
    graphing_columns = c("method1", "method2"),
    column_weights = "value",
    sorting_algorithm = "tsp",
    column_sorting_algorithm = "tsp",
```

```
    coloring_algorithm = "right"
)
```

`plot_alluvial-internals` *wompwomp alluvial plotting*

Description

plot_alluvial() and its helpers: build an alluvial diagram from a data frame with prep_for_lodes() + sort_to_uncross(), colour it, and render with ggalluvial. (Merged from the former biowomp package.)

`prep_for_lodes` *Preprocess data*

Description

Preprocess data (load in, add integer columns, reorder columns to match cols, and group as needed)

Usage

```
prep_for_lodes(
  data,
  cols,
  wt = NULL,
  default_sorting = "alphabetical",
  missing_value = "Missing",
  verbose = FALSE,
  print_params = FALSE,
  do_gather_set_data = FALSE,
  color_band_column = NULL,
  do_add_int_columns = FALSE
)
```

Arguments

`data`	A data frame, tibble, or CSV file path. Must be in one of two formats: (1) wt == NULL: Each row represents an entity, each column represents a grouping, and each entry represents the membership of the entity in that row to the grouping in that column. Must contain at least two columns (two cols). (2) wt != NULL: Each row represents a combination of groupings, each column from `cols` represents a grouping, and the column `wt` represents the number of entities in that combination of groupings. Must contain at least three columns (two `cols`, one `wt`).

`cols` Character vector. Vector of column names from `data` to be used in graphing (i.e., alluvial plotting).

`wt` Optional character. Column name from `data` that contains the weights of each combination of groupings if `data` is in format (2) (see above).

`default_sorting` Character. Default column sorting in `prep_for_lodes()` if integer columns do not exist. Options are 'alphabetical' (default), 'reverse_alphabetical', 'increasing', 'decreasing', 'random'.

`missing_value` Character or `NA`. Missing (`NA`) entries of `cols` are kept as a stratum of their own, and this is the label they are given (default `"Missing"`). If a column that has missing entries already uses the label as a value, the two are merged into one stratum, with a warning; choose a label the data do not use to keep them apart. `NA` keeps the entries missing instead of labeling them: the columns are returned as factors with an explicit `NA` level (as from `forcats::fct_na_value_to_level()`), which cannot collide with any value.

`verbose` Logical. If TRUE, will display messages during the function.

`print_params` Logical. If TRUE, will print function params.

`do_gather_set_data` Internal flag; not recommended to modify.

`color_band_column` Internal flag; not recommended to modify.

`do_add_int_columns` Internal flag; not recommended to modify.

Value

A data frame where each row represents a combination of groupings, each column from `cols` represents a grouping, and the column `wt` ('value' if `wt` == NULL) represents the number of entities in that combination of groupings. For each column in `cols`, there will be an additional column `col1_int`, `col2_int`, etc. where each column corresponds to a position mapping of groupings in the respective entry of `cols` - for example, `col1_int` corresponds to `cols[1]`, `col2_int` corresponds to `cols[2]`, etc.

Examples

```
set.seed(235488)
data <- data.frame(
  method1 = sample(1:3, 100, TRUE),
  method2 = sample(4:6, 100, TRUE)
)
head(data)
lapply(data, unique)

# Example 1: data format 1
clus_df_gather <- prep_for_lodes(
  data,
  cols = c("method1", "method2")
)
```

```
print(clus_df_gather)
lapply(clus_df_gather[, 1:2], levels)

# Example 2: data format 2
clus_df_gather <- data |>
    dplyr::mutate_if(
      is.numeric,
      function(x) factor(x, levels = as.character(sort(unique(x))))
    ) |>
    dplyr::group_by_all() |>
    dplyr::count(name = "value")
print(clus_df_gather)
lapply(clus_df_gather[, 1:2], unique)
clus_df_gather <- prep_for_lodes(
  clus_df_gather,
  cols = c("method1", "method2"),
  wt = "value"
)
print(clus_df_gather)
lapply(clus_df_gather[, 1:2], unique)
```

`sort_to_uncross` *Sorts a dataframe to minimize crossings in a parallel sets / alluvial plot.*

Description

Sorts a dataframe with the algorithm specified by `method`.

Usage

```
sort_to_uncross(
  data,
  cols,
  wt = NULL,
 method = c("neighbornet", "tsp", "greedy", "barycenter", "median", "none", "random"),
  column_method = c("neighbornet", "tsp", "none", "random"),
  alpha = 2,
  beta = alpha,
  fixed_column = NULL,
  verbose = FALSE,
  options = NULL
)
```

Arguments

`data`	A data frame/tibble. Must be in one of two formats: (1) wt == NULL: Each row represents an entity, each column represents a grouping, and each entry represents the membership of the entity in that row to the grouping in that column.

Must contain at least two columns (two cols). (2) wt != NULL: Each row represents a combination of groupings, each column from `cols` represents a grouping, and the column `wt` represents the number of entities in that combination of groupings. Must contain at least three columns (two `cols`, one `wt`).

`cols` Character vector. Vector of column names from `data` to be used in graphing (i.e., alluvial plotting).

`wt` Optional character. Column name from `data` that contains the weights of each combination of groupings if `data` is in format (2) (see above).

`method` Character. Algorithm with which to sort the values in the dataframe. Can choose from: 'neighbornet' (default), 'tsp', 'greedy', 'barycenter', 'median', 'random', 'none'. All of them accept any number of `cols` and honor `fixed_column`. 'neighbornet' builds the block distance matrix and orders the blocks with the NeighborNet algorithm (the W_POMP method described in the paper). 'tsp' is identical except the block ordering is produced by the Traveling Salesman Problem solver from the TSP package rather than NeighborNet. 'greedy', 'barycenter' and 'median' are layer-by-layer sweeps: one pass forward across the axes and one back, each pass reordering an axis against a neighboring axis. 'greedy' places each block as close as possible to its heaviest neighbor in the neighboring axis (O(n_i * n_j) per pass); 'barycenter' and 'median' are the classic Sugiyama-style heuristics that place each block at the weighted mean or weighted median position of its neighbors (O(a log a) per pass, for a alluvia), typically at the cost of noisier (less minimized) crossing counts. 'random' randomly maps blocks. 'none' keeps the mappings as-is when passed into the function. The former names 'greedy_wolf', 'greedy_wblf', 'barycenter_one_sided' and 'median_one_sided' are deprecated: 'greedy_wblf' is 'greedy', and the others are 'greedy', 'barycenter' or 'median' with `fixed_column` (defaulting to `cols[1]`).

`column_method` Character. Algorithm to use for determining column order. Options are 'neighbornet' (default), 'tsp', 'random', and 'none'. Only applies when `cols` has more than two entries. With 'greedy', 'barycenter' and 'median', the strata are swept once in the given axis order, the axis order is chosen from that result, and the strata are swept again in the new order.

`alpha` Positive number (default 2). Ratio $d_{\max}/c$ between the distance assigned to two blocks in different axes that share no observations and the scale c of the $-\log$(edge weight) distances between blocks that do. Larger values hold unconnected blocks further apart relative to the spread induced by differences in edge weight. Together with `beta` this is the only tuning knob of the block distance matrix: the NeighborNet (and TSP) cycle is unchanged when the whole matrix is rescaled, so the absolute values c, $d_{\max}$ and d_{same} matter only through the ratios `alpha` and `beta` (see `weight_scalar`, `matrix_initialization_value` and `same_side_matrix_initialization_value` in `sort_to_uncross_options()` for the underlying constants). Only applies when `method %in% c('neighbornet', 'tsp')`.

`beta` Positive number (default `alpha`). Ratio d_{same}/c between the distance assigned to two distinct blocks of the same axis and the scale c of the edge-weight distances. Only applies when `method %in% c('neighbornet', 'tsp')`.

`fixed_column` Optional character or integer vector. Names, or positions in `cols`, of the axes whose stratum order is kept as it comes in (the order `method = 'none'` would re-

turn, set by default_sorting in sort_to_uncross_options() unless the data already carry col*_int columns); only the other axes are sorted. With 'greedy', 'barycenter' and 'median', order propagates outward from the fixed axes. With 'neighbornet' and 'tsp', which order a distance matrix rather than axes, the fixed order is encouraged in the distance matrix and then imposed exactly on the resulting cycle, whose start and direction are chosen to minimize crossings with the fixed axes in place. Because the cycle is only nudged toward the fixed order, 'greedy', 'barycenter' and 'median' usually reach fewer crossings than 'neighbornet' and 'tsp' when axes are fixed. NULL (default) sorts every axis. Does not affect the order of the axes themselves (see column_method).

verbose — Logical. If TRUE, will display messages during the function.

options — Additional arguments. See sort_to_uncross_options().

Value

A data frame where each row represents an alluvium and each column represents an axis. Each column of cols represents an axis, stored as a factor ordered in ascending order of strata. There is an additional column wt ('value' if NULL) that represents the size of the alluvium in that row. The order of columns represents the recommended order of axes.

Examples

```
# Example 1: data format 1 (uncounted)
set.seed(429144)
data <- data.frame(
  method1 = factor(LETTERS[sample(1:3, 100, TRUE)]),
  method2 = factor(LETTERS[27 - sample(1:3, 100, TRUE)])
)
head(data)
lapply(data, levels)
clus_df_gather <- sort_to_uncross(
  data,
  cols = c("method1", "method2"),
  method = "tsp",
  column_method = "tsp"
)
print(clus_df_gather)
lapply(clus_df_gather[, 1:2], levels)

# Example 2: data format 2 (counted)
set.seed(806949)
data <- data.frame(
  method1 = factor(LETTERS[sample(1:3, 100, TRUE)]),
  method2 = factor(LETTERS[27 - sample(1:3, 100, TRUE)])
)
clus_df_gather <- data |>
  dplyr::mutate_if(
    is.numeric,
    function(x) factor(x, levels = as.character(sort(unique(x))))
  ) |>
  dplyr::group_by_all() |>
```

```
  dplyr::count(name = "value")
print(clus_df_gather)
lapply(clus_df_gather[, 1:2], levels)
clus_df_gather <- sort_to_uncross(
  clus_df_gather,
  cols = c("method1", "method2"),
  wt = "value",
  method = "tsp",
  column_method = "tsp"
)
print(clus_df_gather)
lapply(clus_df_gather[, 1:2], levels)
```

`sort_to_uncross_options` *Control Options for* `sort_to_uncross()`

Description

Creates a list of control parameters that modify the behavior of `sort_to_uncross()`. These options allow tuning algorithmic behavior without cluttering the main function arguments.

Usage

```
sort_to_uncross_options(
  optimize_column_order_per_cycle = FALSE,
  weight_scalar = 5e+05,
  matrix_initialization_value = NULL,
  same_side_matrix_initialization_value = NULL,
  matrix_initialization_value_column_order = 1e+06,
  weight_scalar_column_order = 1,
  column_metric = c("edge_crossing", "ari"),
  weighted_metric = TRUE,
  cycle_start_positions = NULL,
  random_initializations = 1,
  preprocess_data = TRUE,
 default_sorting = c("alphabetical", "reverse_alphabetical", "increasing", "decreasing",
    "random", "fixed"),
  missing_value = "Missing",
  print_params = FALSE,
  do_compute_alluvial_statistics = FALSE
)
```

Arguments

`optimize_column_order_per_cycle`
: Logical. If TRUE, will optimize the order of `cols` to minimize edge overlap upon each cycle. If FALSE, will optimize the order of `cols` to minimize

edge overlap on the beginning cycle only. Only applies when `method %in% c('neighbornet', 'tsp')` and `length(cols) > 2`.

`weight_scalar` Advanced. Positive number c by which $-\log(\text{edge weight})$ is multiplied in the block distance matrix. Because the NeighborNet cycle is invariant under rescaling of the whole matrix (up to floating-point ties), `weight_scalar` only sets the absolute scale of the matrix; the ratios that actually shape the cycle are `alpha` and `beta` in `sort_to_uncross()`. Only applies when `method %in% c('neighbornet', 'tsp')`.

`matrix_initialization_value`
Advanced. Positive number $d_{\max}$: the distance placed between two blocks in different axes that share no observations. `NULL` (default) derives it as `alpha * weight_scalar`; supplying a value overrides `alpha`. Only applies when `method %in% c('neighbornet', 'tsp')`.

`same_side_matrix_initialization_value`
Advanced. Positive number d_{same}: the distance placed between two distinct blocks of the same axis. `NULL` (default) derives it as `beta * weight_scalar`; supplying a value overrides `beta`. Only applies when `method %in% c('neighbornet', 'tsp')`.

`matrix_initialization_value_column_order`
Positive integer. Initialized value in distance matrix for optimizing column order. Only applies when `column_method != 'none'`.

`weight_scalar_column_order`
Positive integer. Scalar with which to loss function after taking their log1p in the distance matrix for optimizing column order. Only applies when `column_method != 'none'`.

`column_metric` Character. Metric to use for determining column order. Options are "edge_crossing" (default) or "ARI". Only applies when `column_method != 'none'`.

`weighted_metric`
Logical. Determines if the objective is total number of edge crossings (weighted_metric=FALSE) or sum of product of overlapping edge weights (weighted_metric=TRUE).

`cycle_start_positions`
Set. Cycle start positions to consider. Anything outside this set will be skipped. Only applies when `method %in% c('neighbornet', 'tsp')`.

`random_initializations`
Integer. Number of initializations of the stratum positions of the non-fixed axes: the first starts from the incoming order and each further one from a random order, and the one with the lowest crossing objective is kept. Only applies when `method %in% c('greedy', 'barycenter', 'median')`.

`preprocess_data`
Logical. If TRUE, will preprocess the data with the `prep_for_lodes()` function.

`default_sorting`
Character. Default column sorting in `prep_for_lodes()` if integer columns do not exist. Options are 'alphabetical' (default), 'reverse_alphabetical', 'increasing', 'decreasing', 'random'.

`missing_value` Character or NA. Missing (NA) entries of `cols` are sorted as a stratum of their own, and this is the label they are given in the result (default `"Missing"`). If a column that has missing entries already uses the label as a value, the two are merged into one stratum, with a warning; choose a label the data do not use to keep them apart. NA keeps the entries missing instead of labeling them: the stratum is returned as an explicit NA factor level (as from `forcats::fct_na_value_to_level()`), which cannot collide with any value. Note that **ggalluvial** does not draw missing strata, so use a label rather than NA for the stratum to appear in the plot. Under `default_sorting`, that level is placed where the string `"NA"` would be. Only applies when `preprocess_data` is TRUE.

`print_params` Logical. If TRUE, will print function params.

`do_compute_alluvial_statistics`
Internal flag; not recommended to modify.

Value

A named list of control parameters, to be passed into `sort_to_uncross()` via the `options` argument.

Examples

```
data <- data.frame(
  method1 = LETTERS[sample(1:3, 100, TRUE)],
  method2 = LETTERS[27 - sample(1:3, 100, TRUE)]
)
opts <- sort_to_uncross_options(
  default_sorting = "reverse_alphabetical",
  weighted_metric = FALSE
)
sort_to_uncross(data = data, cols = c('method1', 'method2'), options = opts)
```

Index

compute_color_agreement, 2
compute_crossing_objective, 3

get_lode_clusters, 5
get_lode_clusters(), *2, 3*
get_lode_clusters_options, 6
get_lode_clusters_options(), *3*

lode_cluster_pal, 7

plot_alluvial, 8
plot_alluvial-internals, 14
prep_for_lodes, 14
prep_for_lodes(), *15*, *20*

sort_to_uncross, 16
sort_to_uncross(), *8*, *10, 11*, *19–21*
sort_to_uncross_options, 19
sort_to_uncross_options(), *17, 18*