Package: wompwomp (via r-universe)

June 29, 2026

Version 0.99.0

Date 2026-06-23

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. Useful for improving visualizations with alluvial plots by ``untangling'' the graphs.

License MIT + file LICENSE

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.3

Imports dplyr, igraph, tibble, rlang, tidyr, purrr, TSP, R6, stats, tidyselect, vctrs, utils

Depends R (>= 4.1.0)

Suggests testthat, vdiffr, magick, knitr, rmarkdown, gtools, mclust, here, stringr, reticulate, ggplot2, ggalluvial, sessioninfo

Config/testthat/edition 3

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

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

NeedsCompilation no

Author Joseph Rich [aut, cre] (ORCID: <https://orcid.org/0000-0003-1400-8479>), Conrad Oakes [aut], Lior Pachter [aut] (ORCID: <https://orcid.org/0000-0002-9164-6231>)

Maintainer Joseph Rich <jmrich@caltech.edu>

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

Date/Publication 2026-06-29 13:50:02 UTC

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

RemoteRef HEAD

RemoteSha c3a2ae810104971b294a8f151e34b5e1fa2b5656

Contents

`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,
  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().
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)
```

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.

Value

A vector of colors.

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)
```

---

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",
  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'.
`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)
```

`setup_python_env` *Setup Python environment for this package*

Description

This function installs Miniconda if needed, creates a dedicated environment, and installs the required Python packages.

Usage

```
setup_python_env(
  envname = "wompwomp_env",
  packages = c(numpy = "numpy", splitspy = "splitspy", pandas = "pandas", scipy =
    "scipy", leidenalg = "leidenalg", igraph = "python-igraph"),
  use_conda = TRUE,
  yes = FALSE
)
```

Arguments

`envname`	Conda environment name
`packages`	Packages to install
`use_conda`	Use conda vs. virtualenv for python environment
`yes`	Logical. Install without prompt

Value

Invisibly returns TRUE if the environment is set up successfully.

Examples

```
## Not run:
setup_python_env()

## End(Not run)
```

`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("tsp", "neighbornet", "greedy_wolf", "greedy_wblf", "none", "random"),
  column_method = c("tsp", "neighbornet", "none", "random"),
  weight_scalar = 5e+05,
  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: 'tsp', 'greedy_wolf', 'greedy_wblf', 'none'. 'tsp' performs Traveling Salesman Problem solver from the TSP package. greedy_wolf' implements a custom greedy algorithm where one layer is fixed, and the other layer is sorted such that each node is positioned as close to its largest parent from the fixed side as possible in a greedy fashion. 'greedy_wblf' implements the 'greedy_wolf' algorithm described previously twice, treating each column as fixed in one iteration and free in the other iteration. 'greedy_wolf' and 'greedy_wblf' are only valid when `cols` has exactly two entries. 'random' randomly maps blocks. 'none' keeps the mappings as-is when passed into the function.
column_method	Character. Algorithm to use for determining column order. Options are 'tsp' (default), 'random', and 'none'.
weight_scalar	Positive integer. Scalar with which to multiply edge weights after taking their -log in the distance matrix for nodes with a nonzero edge. Only applies when `method == 'tsp'`.
fixed_column	Character or Integer. Name or position of the column in `cols` to keep fixed during sorting. Only applies when `method == 'greedy_wolf'`.
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,
  matrix_initialization_value = 1e+06,
  same_side_matrix_initialization_value = 1e+06,
  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"),
  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 == 'tsp'` and `length(cols) > 2`.

`matrix_initialization_value`
: Positive integer. Initialized value in distance matrix for nodes in different layers without a shared edge/path. Only applies when `method == 'tsp'`.

`same_side_matrix_initialization_value`
: Positive integer. Initialized value in distance matrix for nodes in the same layer. Only applies when `method == '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 == 'tsp'`.

`random_initializations`
: Integer. Number of random initializations for the positions of each grouping in `cols`. Only applies when `method == 'greedy_wolf'` or `method == 'greedy_wblf'`.

`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'.

`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",
  matrix_initialization_value = 100
)
sort_to_uncross(data = data, cols = c('method1', 'method2'), options = opts)
```

Index