Package 'chessboard'

October 14, 2023

Type Package

Title Create Network Connections Based on Chess Moves

Version 0.1

Description Provides functions to work with directed (asymmetric) and undirected (symmetric) spatial networks. It makes the creation of connectivity matrices easier, i.e. a binary matrix of dimension n x n, where n is the number of nodes (sampling units) indicating the presence (1) or the absence (0) of an edge (link) between pairs of nodes. Different network objects can be produced by 'chessboard': node list, neighbor list, edge list, connectivity matrix. It can also produce objects that will be used later in Moran's Eigenvector Maps (Dray et al. (2006) <doi:10.1016/j.ecolmodel.2006.02.015>) and Asymetric Eigenvector Maps (Blanchet et al. (2008) <doi:10.1016/j.ecolmodel.2008.04.001>), methods available in the package 'adespatial' (Dray et al. (2023) <https://CRAN.R-project.org/package=adespatial>). This work is part of the FRB-CESAB working group Bridge <https://www.fondationbiodiversite.fr/en/the-frb-in-action/programs-and-projects/le-cesab/bridge/>.

URL https://github.com/frbcesab/chessboard

BugReports https://github.com/frbcesab/chessboard/issues

License GPL (>= 2)

Encoding UTF-8

Imports dplyr, ggplot2, magrittr, rlang, sf, tidyr

Suggests knitr, igraph, patchwork, rmarkdown, testthat (>= 3.0.0)

RoxygenNote 7.2.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Nicolas Casajus [aut, cre, cph] (<https://orcid.org/0000-0002-5537-5294>), Erica Rievrs Borges [aut] (<https://orcid.org/0000-0001-7751-6265>),

Eric Tabacchi [aut] (<https://orcid.org/0000-0001-7729-4439>),
Guillaume Fried [aut] (<https://orcid.org/0000-0002-3653-195X>),
Nicolas Mouquet [aut] (<https://orcid.org/0000-0003-1840-6984>)

Maintainer Nicolas Casajus <nicolas.casajus@fondationbiodiversite.fr>

Repository CRAN

Date/Publication 2023-10-14 08:00:02 UTC

R topics documented:

append_edge_lists *Append several edge lists*

Description

Appends several edge lists created by `create_edge_list()`. Merged edges will be ordered and duplicates will be removed.

Usage

```
append_edge_lists(...)
```

Arguments

`...`	one or several edge lists `data.frame`. Outputs of the function `create_edge_list()`.

Value

A `data.frame` with `n` rows (where `n` is the total number of edges) and the following two columns:

- `from`: the node label of one of the two endpoints of the edge
- `to`: the node label of the other endpoint of the edge

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

edges_1 <- create_edge_list(nodes, method = "pawn", directed = TRUE)
edges_2 <- create_edge_list(nodes, method = "bishop", directed = TRUE)

edges <- append_edge_lists(edges_1, edges_2)
```

append_matrix *Combine several connectivity matrices*

Description

Combines different connectivity matrices by row names and column names by performing a 2-dimensional full join. Missing edges are filled with `0` (default) or `NA` (argument `na_to_zero`).

Usage

```
append_matrix(..., na_to_zero = TRUE)
```

Arguments

`...`	one or several `matrix` objects created by `connectivity_matrix()`.
`na_to_zero`	a `logical` value. If `TRUE` (default) missing edges are coded as `0`. Otherwise they will be coded as `NA`.

Value

A connectivity matrix of dimensions `n x n`, where `n` is the total number of unique nodes across all provided matrices.

Examples

```
mat1 <- matrix(rep(1, 9), nrow = 3)
colnames(mat1) <- c("A", "B", "C")
rownames(mat1) <- c("A", "B", "C")
mat1

mat2 <- matrix(rep(1, 9), nrow = 3)
colnames(mat2) <- c("D", "E", "F")
rownames(mat2) <- c("D", "E", "F")
mat2

mat3 <- matrix(rep(1, 9), nrow = 3)
colnames(mat3) <- c("F", "G", "H")
rownames(mat3) <- c("F", "G", "H")
mat3

append_matrix(mat1, mat2, mat3)

append_matrix(mat1, mat2, mat3, na_to_zero = FALSE)
```

bishop *Find neighbors according to bishop movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the bishop movement. This movement is derived from the chess game. The bishop can move along the two diagonals.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
bishop(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
  self = FALSE
)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the `nodes` (`data.frame`) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- bishop(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- bishop(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- bishop(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- bishop(nodes, focus, degree = 3, directed = TRUE,
                    reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`bishop_left` *Find neighbors according to bishop left movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the bishop left movement. This movement is derived from the `bishop()` method and can only move along the bottom-right to top-left diagonal.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
bishop_left(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
  self = FALSE
)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the `nodes` (`data.frame`) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
```

```
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- bishop_left(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- bishop_left(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- bishop_left(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- bishop_left(nodes, focus, degree = 3, directed = TRUE,
                         reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`bishop_right` *Find neighbors according to bishop right movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the bishop right movement. This movement is derived from the `bishop()` method and can only move along the bottom-left to top-right diagonal.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
bishop_right(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
```

```
  reverse = FALSE,
  self = FALSE
)
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output of the function create_node_labels().
focus	an character of length 1. The node label for which the neighbors must be found. Must exist in the nodes object.
degree	an integer of length 1. The maximum number of neighbors to search for.
directed	a logical of length 1. If FALSE (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
reverse	a logical of length 1. If TRUE, change the default orientation of the network. This argument is ignored if directed = FALSE. See examples for further detail.
self	a logical of length 1. If TRUE, a node can be its own neighbor. Default is FALSE.

Details

This function is internally called by create_edge_list() but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (focus).

Value

A subset of the nodes (data.frame) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- bishop_right(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

```
# Higher degree of neighborhood ----
neighbors <- bishop_right(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- bishop_right(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- bishop_right(nodes, focus, degree = 3, directed = TRUE,
                          reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`connectivity_matrix` *Create a connectivity matrix from an edge list*

Description

Converts an edge list to an connectivity matrix (also known as adjacency matrix).

Usage

```
connectivity_matrix(
  edges,
  lower = TRUE,
  upper = TRUE,
  diag = TRUE,
  na_to_zero = TRUE
)
```

Arguments

`edges`	a `data.frame` with the following two columns: `from` (the first node of the edge) and `to` (the second node of the edge). The output of the functions `create_edge_list()` or `append_edge_lists()`.
`lower`	a `logical` value. If `TRUE` (default), keep values in the lower triangle of the matrix. Otherwise they will be replaced by `NA` (or `0`).
`upper`	a `logical` value. If `TRUE` (default), keep values in the upper triangle of the matrix. Otherwise they will be replaced by `NA` (or `0`).
`diag`	a `logical` value. If `TRUE` (default), keep values in the diagonal of the matrix. Otherwise they will be replaced by `NA` (or `0`).
`na_to_zero`	a `logical` value. If `TRUE` (default), missing edges are coded as `0`. Otherwise they will be coded as `NA`.

Value

A connectivity matrix of dimensions `n x n`, where `n` is the number of nodes.

Examples

```
# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites  <- read.csv(path_to_file)

# Select first location ----
adour_sites <- adour_sites[adour_sites$"location" == 1, ]

# Create node labels ----
adour_nodes <- create_node_labels(data     = adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat = "quadrat")

# Find edges with 1 degree of neighborhood (pawn method) ----
adour_edges <- create_edge_list(adour_nodes, method = "pawn",
                                directed = TRUE)

# Get connectivity matrix ----
connectivity_matrix(adour_edges)

# Get connectivity matrix ----
connectivity_matrix(adour_edges, na_to_zero = FALSE)
```

`create_edge_list` *Create an edge list*

Description

Creates a list of edges (links) between nodes (sampling units) based on the detection of neighbors and according to three neighborhood rules:

1. **Degree of neighborhood** (argument `degree`): the number of adjacent nodes that will be used to create **direct** edges. If `degree = 1`, only nodes directly adjacent to the focal node will be considered as neighbors.
2. **Orientation of neighborhood** (argument `method`): can neighbors be detecting horizontally and/or vertically and/or diagonally? The package `chessboard` implements all possible orientations derived from the chess game.
3. **Direction of neighborhood** (arguments `directed` and `reverse`): does the sampling design has a direction? If so (`directed = TRUE`), the network will be considered as **directed** and the direction will follow the order of node labels in both axes (except if `reverse = TRUE`).

It's important to note that, even the package chessboard is designed to deal with spatial networks, this function does not explicitly use spatial coordinates to detect neighbors. Instead it uses the **node labels**. The function create_node_labels() must be used before this function to create node labels.

Usage

```
create_edge_list(
  nodes,
  method,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
  self = FALSE
)
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output of the function create_node_labels().
method	a character of length 1. The method used to detect neighbors. One among 'pawn', 'fool', 'rook', 'bishop', 'bishop_left', 'bishop_right', 'knight', 'knight_left', 'knight_right', 'queen', 'wizard'. For further information, see the functions of the same name (i.e. pawn(), rook(), etc.).
degree	an integer of length 1. The maximum number of neighbors to search for.
directed	a logical of length 1. If FALSE (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
reverse	a logical of length 1. If TRUE, change the default orientation of the network. This argument is ignored if directed = FALSE. See examples for further detail.
self	a logical of length 1. If TRUE, a node can be its own neighbor. Default is FALSE.

Value

A data.frame with n rows (where n is the number of edges) and the following two columns:

- from: the node label of one of the two endpoints of the edge
- to: the node label of the other endpoint of the edge

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)
```

```
nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

edges <- create_edge_list(nodes, method = "pawn", directed = TRUE)
edges

edges <- create_edge_list(nodes, method = "bishop", directed = TRUE)
edges
```

`create_node_labels` *Create unique node labels*

Description

Creates unique node (sampling units) labels in directed (or undirected) spatial (or not) networks.

It's important to note that, even the package `chessboard` is designed to deal with spatial networks, it does not explicitly use spatial coordinates. Every functions of the package will use the **node labels**.

To work, the package `chessboard` requires that the sampling has two dimensions: one from bottom to top (called **quadrats**), and one from left to right (called **transects**). If the sampling has been conducted along one single dimension (**transects** or **quadrats**), this function will create a fictitious label for the missing dimension. In other words, the package `chessboard` can work with sampling designs such as regular grids (two dimensions), transects (one dimension), and quadrats (one dimension).

In addition, the package can also deal with multiple locations. In that case, users will need to use the argument `location`.

The node labels will be of the form: `1-2`, where `1` is the identifier of the transect (created by the function if missing), and `2`, the identifier of the quadrat (created by the function if missing).

Usage

```
create_node_labels(data, location, transect, quadrat)
```

Arguments

`data`	a `data.frame` with at least one column, `'transect'` or `'quadrat'`. If only one column is provided and `transect` or `quadrat` is `NULL`, the network will be considered as one-dimensional. If `data` contains both `'transect'` and `'quadrat'` columns, the network will be considered as two-dimensional. The `data.frame` can contain additional columns.
`location`	a `character` of length 1. The name of the column that contains location identifiers. If missing (or `NULL`), a unique location identifier will be created and named `1` (for the purpose of the package only). This argument is optional if the sampling ha been conducted at one location, but required if the survey is structured in multiple locations.

transect — a character of length 1. The name of the column that contains transect identifiers. If missing (or NULL), a unique transect identifier will be created and named 1 (for the purpose of the package only). If missing, the network will be considered as one-dimensional.

quadrat — a character of length 1. The name of the column that contains quadrat identifiers. If missing (or NULL), a unique quadrat identifier will be created and named 1 (for the purpose of the package only). If missing, the network will be considered as one-dimensional.

Value

A data.frame with at least the four following columns:

- node, the node label
- location, the identifier of the location
- transect, the identifier of the transect
- quadrat, the identifier of the quadrat Other columns present in the original dataset will also be added.

Examples

```
library("chessboard")

# Two-dimensional sampling ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)
sites_infos

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")
nodes

gg_chessboard(nodes)

# One-dimensional sampling (only transects) ----
transects_only <- data.frame("transect" = 1:5)

nodes <- create_node_labels(transects_only,
                            transect = "transect")
nodes

gg_chessboard(nodes)

# One-dimensional sampling (only quadrats) ----
quadrats_only <- data.frame("quadrat" = 1:5)

nodes <- create_node_labels(quadrats_only,
                            quadrat = "quadrat")
nodes

gg_chessboard(nodes)
```

`distance_euclidean` *Compute the pairwise Euclidean distance*

Description

Computes the Euclidean distance between two nodes using the function `sf::st_distance()`. If the CRS is not a Cartesian system, the Great Circle distance will be used instead.

Usage

```
distance_euclidean(sites, ...)
```

Arguments

`sites`	an `sf` object of type `POINT`. A spatial object containing coordinates of sites. Note that the first column must be the node label created by the function `create_node_labels()`.
`...`	other argument to pass to `sf::st_distance()`.

Value

A three-column `data.frame` with:

- `from`, the first node
- `to`, the second node
- `weight`, the Euclidean distance between the two nodes

Examples

```
# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites <- read.csv(path_to_file)

# Select the 15 first sites ----
adour_sites <- adour_sites[1:15, ]

# Create node labels ----
adour_sites <- create_node_labels(adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat  = "quadrat")

# Convert sites to sf object (POINTS) ----
adour_sites <- sf::st_as_sf(adour_sites, coords = c("longitude", "latitude"),
                            crs = "epsg:2154")

# Compute distances between pairs of sites ----
weights <- distance_euclidean(adour_sites)

head(weights)
```

edges_to_sf *Convert edge list to spatial object*

Description

Converts an edge list to an sf spatial object of type LINESTRING with one row per edge.

Usage

```
edges_to_sf(edges, sites)
```

Arguments

edges	a data.frame with the following two columns: from (the first node of the edge) and to (the second node of the edge). The output of the functions create_edge_list() or append_edge_lists().
sites	an sf object of type POINT. A spatial object with coordinates of sites (nodes). Note that the **first column** must be the node labels.

Value

An sf spatial object of type LINESTRING where the number of rows correspond to the number of edges.

Examples

```
# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites  <- read.csv(path_to_file)

# Select first location ----
adour_sites <- adour_sites[adour_sites$"location" == 1, ]

# Create node labels ----
adour_nodes <- create_node_labels(data     = adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat = "quadrat")

# Find edges with 1 degree of neighborhood (pawn method) ----
adour_edges <- create_edge_list(adour_nodes, method = "pawn",
                                directed = TRUE)

# Convert sites to spatial POINT ----
adour_sites_sf <- sf::st_as_sf(adour_nodes, coords = 5:6, crs = "epsg:2154")

# Convert edges to spatial LINESTRING ----
edges_sf <- edges_to_sf(adour_edges, adour_sites_sf)
```

```
head(edges_sf)

# Visualization ----
plot(sf::st_geometry(adour_sites_sf), pch = 19)
plot(sf::st_geometry(edges_sf), add = TRUE)


# Find edges with 1 degree of neighborhood (pawn and bishop methods) ----
adour_edges_1 <- create_edge_list(adour_nodes, method = "pawn",
                                  directed = TRUE)
adour_edges_2 <- create_edge_list(adour_nodes, method = "bishop",
                                  directed = TRUE)

# Append edges ----
adour_edges <- append_edge_lists(adour_edges_1, adour_edges_2)

# Convert sites to spatial POINT ----
adour_sites_sf <- sf::st_as_sf(adour_nodes, coords = 5:6, crs = "epsg:2154")

# Convert edges to spatial LINESTRING ----
edges_sf <- edges_to_sf(adour_edges, adour_sites_sf)
head(edges_sf)

# Visualization ----
plot(sf::st_geometry(adour_sites_sf), pch = 19)
plot(sf::st_geometry(edges_sf), add = TRUE)
```

`edges_weights_matrix` *Create an edges weights matrix*

Description

Creates an edges weights matrix from the output of `distance_euclidean()`.

Usage

```
edges_weights_matrix(distances, lower = TRUE, upper = TRUE, diag = TRUE)
```

Arguments

`distances`	a `data.frame` with the following three columns: `from` (the first node of the edge), `to` (the second node of the edge), and `weight` (the weight of the edge between the two nodes, e.g. a distance).
`lower`	a `logical` value. If `TRUE` (default), keep values in the lower triangle of the matrix. Otherwise they will be replaced by `NA`.
`upper`	a `logical` value. If `TRUE` (default), keep values in the upper triangle of the matrix. Otherwise they will be replaced by `NA`.
`diag`	a `logical` value. If `TRUE` (default), keep values in the diagonal of the matrix. Otherwise they will be replaced by `NA`.

Value

An edges weights `matrix` of dimensions `n x n`, where `n` is the number of nodes (sites).

Examples

```
# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites <- read.csv(path_to_file)

# Select the 15 first sites ----
adour_sites <- adour_sites[1:15, ]

# Create node labels ----
adour_sites <- create_node_labels(adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat  = "quadrat")

# Convert sites to sf object (POINTS) ----
adour_sites_sf <- sf::st_as_sf(adour_sites,
                               coords = c("longitude", "latitude"),
                               crs = "epsg:2154")

# Compute distances between pairs of sites along the Adour river ----
adour_dists <- distance_euclidean(adour_sites_sf)

# Create Edges weights matrix ----
edges_weights_matrix(adour_dists)

# Create Edges weights matrix (with options) ----
edges_weights_matrix(adour_dists, lower = FALSE)
edges_weights_matrix(adour_dists, upper = FALSE)
edges_weights_matrix(adour_dists, diag = FALSE)
```

---

edges_weights_vector *Create an edges weights vector*

Description

Creates an edges weights vector that can be used in `aem()` of the package `adespatial`. Resulting edges weights equal to 0 will be replaced by `4 x max(w)`, where `max(w)` is the maximal weight in the matrix.

Usage

```
edges_weights_vector(x, y)
```

Arguments

x	a list of length 2. The nodes_by_edges matrix returned by nodes_by_edges_matrix() (or aem.build.binary() of the package adespatial).
y	a data.frame with the following three columns: from (the first node of the edge), to (the second node of the edge), and weight (the weight of the edge between the two nodes, e.g. a distance).

Value

An edges weights vector.

Examples

```
# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites <- read.csv(path_to_file)

# Select the 15 first sites ----
adour_sites <- adour_sites[1:15, ]

# Create node labels ----
adour_sites <- create_node_labels(adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat  = "quadrat")

# Convert sites to sf object (POINTS) ----
adour_sites_sf <- sf::st_as_sf(adour_sites,
                               coords = c("longitude", "latitude"),
                               crs = "epsg:2154")

# Create edges based on the pawn move (directed network) ----
adour_edges <- create_edge_list(adour_sites, method = "pawn",
                                directed = TRUE)

# Create nodes-by-edges matrix ----
adour_matrix <- nodes_by_edges_matrix(adour_edges)

# Compute Euclidean distances between pairs of sites ----
adour_dists <- distance_euclidean(adour_sites_sf)

# Create Edges weights vector ----
edges_weights_vector(adour_matrix, adour_dists)
```

---

fool *Find neighbors according to fool movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the fool movement. This movement is derived from the chess game. The fool can only move horizontally, i.e. through a **quadrat**.

The detection of neighbors using the fool method can work with two-dimensional sampling (both **transects** and **quadrats**) and one-dimensional sampling of type **transects-only**. For sampling of type **quadrats-only**, please use the function `pawn()`.

Usage

```
fool(nodes, focus, degree = 1, directed = FALSE, reverse = FALSE, self = FALSE)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the `nodes` (`data.frame`) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")
```

```
focus     <- "5-5"

# Default settings ----
neighbors <- fool(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- fool(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- fool(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- fool(nodes, focus, degree = 3, directed = TRUE, reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`geom_edges` *Link neighbors by arrow on a chessboard*

Description

Links neighbors (cells) on a chessboard plotted with `gg_chessboard()`.

Usage

```
geom_edges(nodes, focus, neighbors)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output if the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label to be emphasized on the chessboard. Must exist in the `nodes` object.
`neighbors`	a `data.frame` with the following at least three columns: `node`, `transect`, and `quadrats`. See `pawn()`, `fool()`, etc. for further information.

Value

A `geom_segment` that must be added to a `ggplot2` object.

Examples

```
library("chessboard")

sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus <- "5-5"

neighbors <- wizard(nodes, focus = focus, degree = 4, directed = FALSE,
                    reverse = TRUE)

gg_chessboard(nodes) +
  geom_neighbors(nodes, neighbors) +
  geom_edges(nodes, focus, neighbors) +
  geom_node(nodes, focus)
```

---

geom_neighbors *Highlight neighbors on a chessboard*

Description

Highlights neighbors (cells) on a chessboard plotted with gg_chessboard().

Usage

```
geom_neighbors(nodes, neighbors)
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output if the function create_node_labels().
neighbors	a data.frame with the following at least three columns: node, transect, and quadrats. See pawn(), fool(), etc. for further information.

Value

A geom_point that must be added to a ggplot2 object.

Examples

```
library("chessboard")

# Two-dimensional sampling ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
```

```
                            transect = "transect",
                            quadrat  = "quadrat")

neighbors <- pawn(nodes, focus = "2-3")

gg_chessboard(nodes) +
  geom_node(nodes, "2-3") +
  geom_neighbors(nodes, neighbors)
```

---

geom_node *Highlight a node on a chessboard*

Description

Highlights a node (cell) on a chessboard plotted with `gg_chessboard()`.

Usage

```
geom_node(nodes, focus)
```

Arguments

nodes	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output if the function `create_node_labels()`.
focus	an `character` of length 1. The node label to be emphasized on the chessboard. Must exist in the `nodes` object.

Value

A list of two `geom_point` that must be added to a `ggplot2` object.

Examples

```
library("chessboard")

# Two-dimensional sampling ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

gg_chessboard(nodes) +
  geom_node(nodes, "2-3")

# One-dimensional sampling (only transects) ----
sites_infos <- data.frame("transect" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
```

```
                            transect = "transect")

gg_chessboard(nodes) +
  geom_node(nodes, "3-1")
```

`get_node_list` *Get the list of nodes*

Description

Retrieves the node list by selecting and ordering the column node of the output of the function `create_node_labels()`.

Usage

```
get_node_list(nodes)
```

Arguments

nodes	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.

Value

A vector of node labels.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")
get_node_list(nodes)
```

gg_chessboard *Plot a sampling as a chessboard*

Description

Plots a sampling as a chessboard of dimensions t x q, with t, the number of transects, and q, the number of quadrats.

Usage

```
gg_chessboard(nodes, xlab = "Transect", ylab = "Quadrat")
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output if the function create_node_labels().
xlab	a character of length 1. The title of the top axis. Default is 'Transect'.
ylab	a character of length 1. The title of the left axis. Default is 'Quadrat'.

Value

A ggplot2 object.

Examples

```
library("chessboard")

# Two-dimensional sampling ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

gg_chessboard(nodes)

# One-dimensional sampling (only transects) ----
sites_infos <- data.frame("transect" = 1:5)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect")

gg_chessboard(nodes)

# One-dimensional sampling (only quadrats) ----
sites_infos <- data.frame("quadrat" = 1:5)

nodes <- create_node_labels(data    = sites_infos,
                            quadrat = "quadrat")
```

```
gg_chessboard(nodes)
```

---

gg_matrix *Plot a connectivity or a nodes-by-edges matrix*

Description

Plots an connectivity or a nodes-by-edges matrix.

Usage

```
gg_matrix(x, title)
```

Arguments

x	a matrix object. An adjacency or nodes-by-edges matrix.
title	a character of length 1. The caption of the figure.

Value

A ggplot2 object.

Examples

```
library("chessboard")

# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites  <- read.csv(path_to_file)

# Select first location ----
#adour_sites <- adour_sites[adour_sites$"location" == 1, ]

# Create node labels ----
adour_nodes <- create_node_labels(data     = adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat = "quadrat")

# Find edges with 1 degree of neighborhood (queen method) ----
adour_edges <- create_edge_list(adour_nodes, method = "queen",
                                directed = FALSE)

# Get connectivity matrix ----
adour_con_matrix <- connectivity_matrix(adour_edges)

# Visualize matrix ----
gg_matrix(adour_con_matrix, title = "Connectivity matrix") +
  ggplot2::theme(axis.text = ggplot2::element_text(size = 6))
```

knight *Find neighbors according to knight movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the knight movement. This movement is derived from the chess game. The knight is the difference between the `wizard()` and the `queen()`.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
knight(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
  self = FALSE
)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the nodes (data.frame) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- knight(nodes, focus, degree = 2)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- knight(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- knight(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- knight(nodes, focus, degree = 3, directed = TRUE,
                    reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

---

knight_left *Find neighbors according to knight left movement*

Description

For one node (argument focus), finds neighbors among a list of nodes according to the knight left movement. This movement is derived from the knight() method.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
knight_left(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
  self = FALSE
)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the `nodes` (`data.frame`) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
```

```
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- knight_left(nodes, focus, degree = 2)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- knight_left(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- knight_left(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- knight_left(nodes, focus, degree = 3, directed = TRUE,
                         reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`knight_right` *Find neighbors according to knight right movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the knight right movement. This movement is derived from the `knight()` method.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
knight_right(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
```

```
  self = FALSE
)
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output of the function create_node_labels().
focus	an character of length 1. The node label for which the neighbors must be found. Must exist in the nodes object.
degree	an integer of length 1. The maximum number of neighbors to search for.
directed	a logical of length 1. If FALSE (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
reverse	a logical of length 1. If TRUE, change the default orientation of the network. This argument is ignored if directed = FALSE. See examples for further detail.
self	a logical of length 1. If TRUE, a node can be its own neighbor. Default is FALSE.

Details

This function is internally called by create_edge_list() but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (focus).

Value

A subset of the nodes (data.frame) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- knight_right(nodes, focus, degree = 2)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
```

```
neighbors <- knight_right(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- knight_right(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- knight_right(nodes, focus, degree = 3, directed = TRUE,
                          reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`matrix_to_edge_list` *Convert an connectivity matrix to an edge list*

Description

Converts a connectivity matrix to an edge list. This function allows to create the same edge list as the one obtained with `create_edge_list()`.

Usage

```
matrix_to_edge_list(x, all = FALSE)
```

Arguments

`x`	a `matrix` object. The connectivity matrix to be converted in an edge list.
`all`	a `logical` value. If `FALSE` (default), removes missing edges.

Value

A `data.frame` with two (or three) columns:

- `from`: label of one of the two nodes of the edge
- `to`: label of the other node of the edge
- `edge`: 0 (no edge) or 1 (edge). This column is returned only if `all = TRUE`.

Examples

```
library("chessboard")

# Two-dimensional sampling ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)
sites_infos

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

edges <- create_edge_list(nodes, method = "pawn", directed = TRUE)

conn_matrix <- connectivity_matrix(edges)

# Convert back to edge list ----
new_edges <- matrix_to_edge_list(conn_matrix)
new_edges

# Check ----
identical(edges, new_edges)
```

`nodes_by_edges_matrix` *Create a nodes-by-edges matrix*

Description

Creates a nodes-by-edges matrix that will be used by `aem()` of the package `adespatial`. This function creates the same output as `aem.build.binary()` of the package `adespatial` but works in a different way: it's only based on node labels (not on coordinates). Also, this function adds labels to nodes and edges.

Usage

```
nodes_by_edges_matrix(edges)
```

Arguments

`edges`	a `data.frame` with the following two columns: `from` (the first node of the edge) and `to` (the second node of the edge).

Value

A list of two elements:

- `se.mat`: the nodes-by-edges matrix of dimensions `n x k`, where `n` is the number of nodes and `k` the number of edges (including the edge between the fictitious origin and the first site);
- `edges`: a `data.frame` of edge list.

Examples

```
library("chessboard")

# Two-dimensional sampling ----
sites_infos <- expand.grid("transect" = 1:3, "quadrat" = 1:5)
sites_infos

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

edges <- create_edge_list(nodes, method = "pawn", directed = TRUE)

# Create nodes-by-edges matrix ----
nodes_by_edges_matrix(edges)
```

---

pawn *Find neighbors according to pawn movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the fool movement. This movement is orthogonal to the `fool()` function. The pawn can only move vertically, i.e. through a **transect**.

The detection of neighbors using the fool method can work with two-dimensional sampling (both **transects** and **quadrats**) and one-dimensional sampling of type **quadrats-only**. For sampling of type **transects-only**, please use the function `fool()`.

Usage

```
pawn(nodes, focus, degree = 1, directed = FALSE, reverse = FALSE, self = FALSE)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the `nodes` (`data.frame`) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- pawn(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- pawn(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- pawn(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- pawn(nodes, focus, degree = 3, directed = TRUE, reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

---

queen *Find neighbors according to queen movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the queen movement. This movement is derived from the chess game. The queen is a combination of the `bishop()` and the `rook()` and can find neighbors horizontally, vertically, and diagonally.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
queen(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
  reverse = FALSE,
  self = FALSE
)
```

Arguments

`nodes`	a `data.frame` with (at least) the following three columns: `node`, `transect`, and `quadrats`. Must be the output of the function `create_node_labels()`.
`focus`	an `character` of length 1. The node label for which the neighbors must be found. Must exist in the `nodes` object.
`degree`	an `integer` of length 1. The maximum number of neighbors to search for.
`directed`	a `logical` of length 1. If `FALSE` (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
`reverse`	a `logical` of length 1. If `TRUE`, change the default orientation of the network. This argument is ignored if `directed = FALSE`. See examples for further detail.
`self`	a `logical` of length 1. If `TRUE`, a node can be its own neighbor. Default is `FALSE`.

Details

This function is internally called by `create_edge_list()` but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (`focus`).

Value

A subset of the `nodes` (`data.frame`) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- queen(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
neighbors <- queen(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- queen(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- queen(nodes, focus, degree = 3, directed = TRUE, reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

---

rook *Find neighbors according to rook movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the rook movement. This movement is derived from the chess game. The rook can move horizontally and vertically. It's a combination of the `pawn()` and the `fool()` functions.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
rook(nodes, focus, degree = 1, directed = FALSE, reverse = FALSE, self = FALSE)
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output of the function create_node_labels().
focus	an character of length 1. The node label for which the neighbors must be found. Must exist in the nodes object.
degree	an integer of length 1. The maximum number of neighbors to search for.
directed	a logical of length 1. If FALSE (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
reverse	a logical of length 1. If TRUE, change the default orientation of the network. This argument is ignored if directed = FALSE. See examples for further detail.
self	a logical of length 1. If TRUE, a node can be its own neighbor. Default is FALSE.

Details

This function is internally called by create_edge_list() but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (focus).

Value

A subset of the nodes (data.frame) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- rook(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Higher degree of neighborhood ----
```

```
neighbors <- rook(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- rook(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- rook(nodes, focus, degree = 3, directed = TRUE, reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

`spatial_weights_matrix` *Create a spatial weights matrix*

Description

Creates a spatial weights matrix by multiplying an adjacency (connectivity) matrix (see `connectivity_matrix()`) and an edges weights matrix (see `edges_weights_matrix()`). Resulting spatial weights equal to 0 will be replaced by `4 x max(w)`, where `max(w)` is the maximal weight in the matrix.

Usage

```
spatial_weights_matrix(x, y)
```

Arguments

`x`	an adjacency `matrix` of dimensions `n x n`, where `n` is the number of nodes (sites). The output of `connectivity_matrix()`.
`y`	an edges weight `matrix` of dimensions `n x n`, where `n` is the number of nodes (sites). The output of `edges_weights_matrix()`.

Value

A spatial weights `matrix` of dimensions `n x n`, where `n` is the number of nodes (sites).

Examples

```
# Import Adour sites ----
path_to_file <- system.file("extdata", "adour_survey_sampling.csv",
                            package = "chessboard")
adour_sites <- read.csv(path_to_file)
```

```
# Select the 15 first sites ----
adour_sites <- adour_sites[1:15, ]

# Create node labels ----
adour_sites <- create_node_labels(adour_sites,
                                  location = "location",
                                  transect = "transect",
                                  quadrat  = "quadrat")

# Create edges based on the pawn move (directed network) ----
adour_edges <- create_edge_list(adour_sites, method = "pawn",
                                directed = TRUE)

# Get connectivity matrix ----
adour_adjacency <- connectivity_matrix(adour_edges)

# Convert sites to sf object (POINTS) ----
adour_sites_sf <- sf::st_as_sf(adour_sites,
                               coords = c("longitude", "latitude"),
                               crs = "epsg:2154")

# Compute distances between pairs of sites along the Adour river ----
adour_dists <- distance_euclidean(adour_sites_sf)

# Create Edges weights matrix ----
adour_weights <- edges_weights_matrix(adour_dists)

# Create Spatial weights matrix ----
spatial_weights_matrix(adour_adjacency, adour_weights)
```

`wizard` *Find neighbors according to wizard movement*

Description

For one node (argument `focus`), finds neighbors among a list of nodes according to the wizard movement. This movement is a combination of the `queen()` and the `knight()` pieces from the chess game and can move in all directions.

The detection of neighbors using this method can only work with two-dimensional sampling (both **transects** and **quadrats**). For sampling of type **transects-only** or **quadrats-only**, please use the functions `fool()` or `pawn()`, respectively.

Usage

```
wizard(
  nodes,
  focus,
  degree = 1,
  directed = FALSE,
```

```
  reverse = FALSE,
  self = FALSE
)
```

Arguments

nodes	a data.frame with (at least) the following three columns: node, transect, and quadrats. Must be the output of the function create_node_labels().
focus	an character of length 1. The node label for which the neighbors must be found. Must exist in the nodes object.
degree	an integer of length 1. The maximum number of neighbors to search for.
directed	a logical of length 1. If FALSE (default), search for neighbors in all directions (undirected network). Otherwise, the network will be considered as directed according to the orientations of the network. The default orientation follows the order of node labels in both axes.
reverse	a logical of length 1. If TRUE, change the default orientation of the network. This argument is ignored if directed = FALSE. See examples for further detail.
self	a logical of length 1. If TRUE, a node can be its own neighbor. Default is FALSE.

Details

This function is internally called by create_edge_list() but it can be directly used to 1) understand the neighbors detection method, and 2) to check detected neighbors for one particular node (focus).

Value

A subset of the nodes (data.frame) where each row is a neighbor of the focal node.

Examples

```
library("chessboard")

# Two-dimensional sampling (only) ----
sites_infos <- expand.grid("transect" = 1:9, "quadrat" = 1:9)

nodes <- create_node_labels(data     = sites_infos,
                            transect = "transect",
                            quadrat  = "quadrat")

focus     <- "5-5"

# Default settings ----
neighbors <- wizard(nodes, focus)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

```
# Higher degree of neighborhood ----
neighbors <- wizard(nodes, focus, degree = 3)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (default orientation) ----
neighbors <- wizard(nodes, focus, degree = 3, directed = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)

# Directed (reverse orientation) ----
neighbors <- wizard(nodes, focus, degree = 3, directed = TRUE,
                    reverse = TRUE)
gg_chessboard(nodes) +
  geom_node(nodes, focus) +
  geom_neighbors(nodes, neighbors)
```

Index