Package: fsr (via r-universe)

September 6, 2024

Type Package

Title Handling Fuzzy Spatial Data

Version 2.0.1.9000

URL https://accarniel.github.io/fsr/, https://github.com/accarniel/fsr

BugReports https://github.com/accarniel/fsr/issues

Depends R (>= 3.6.0)

Imports rlang (>= 0.4.11), methods (>= 2.0.0), sf (>= 1.0.15), dplyr (>= 1.0.6), ggplot2 (>= 3.3.5), stringr (>= 1.4.0), tibble (>= 3.0.1), pso (>= 1.0.3), e1071 (>= 1.7.3)

Suggests lwgeom (>= 0.2.6)

Description Support for fuzzy spatial objects, their operations, and fuzzy spatial inference models based on Spatial Plateau Algebra. It employs fuzzy set theory and fuzzy logic as foundation to deal with spatial fuzziness. It mainly implements underlying concepts defined in the following research papers: (i) ``Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types" <doi:10.1109/FUZZ-IEEE.2018.8491565>; (ii) ``A Systematic Approach to Creating Fuzzy Region Objects from Real Spatial Data Sets" <doi:10.1109/FUZZ-IEEE.2019.8858878>; (iii) ``Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions" <doi:10.1109/FUZZ48607.2020.9177620>; (iv) ``Fuzzy Inference on Fuzzy Spatial Objects (FIFUS) for Spatial Decision Support Systems" <doi:10.1109/FUZZ-IEEE.2017.8015707>; (v) ``Evaluating Region Inference Methods by Using Fuzzy Spatial Inference Models" <doi:10.1109/FUZZ-IEEE55066.2022.9882658>.

License GPL-3

LazyData true

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

NeedsCompilation no

Encoding UTF-8

Collate 'data_types.R' 'internal_functions.R' 'basic_functions.R' 'construction_module.R' 'general_operations.R' 'fuzzy_numerical_operations.R' 'fuzzy_geom_set_operations.R' 'fuzzy_topological_relations.R' 'fsi_module.R'

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

RemoteUrl https://github.com/accarniel/fsr

RemoteRef HEAD

RemoteSha f86fffd974c7c728b44dadebfa0f59f3b0a4874e

Contents

`as_tibble.pgeometry` *Convert a* `pgeometry` *object into tabular data (*`data.frame` *or* `tibble`*)*

Description

These functions convert a `pgeometry` object into a tabular format, such as a `tibble` or `data.frame` object, where the components of the `pgeometry` object compose the rows of the table.

Usage

```
## S3 method for class 'pgeometry'
as_tibble(x, ...)

## S3 method for class 'pgeometry'
as.data.frame(x, ...)
```

Arguments

`x`	A `pgeometry` object.
`...`	<dynamic-dots> Unused.

Details

These functions are S3 methods for `pgeometry`. The `as_tibble()` function converts a `pgeometry` object into a `tibble`, which is a data frame with class `tbl_df`. This allows us to get the internal components of the `pgeometry` object (i.e., spatial features objects and membership degrees) as a data frame with two separate columns: (i) `geometry` (an `sfc` object) and (ii) `md` (*membership degree*). Therefore, each row of this tibble represents a component of the original `pgeometry` object.

It is also possible to call the S3 method `as.data.frame()` to convert a `pgeometry` object into a `data.frame` object.

Value

A tabular object (`data.frame` or `tibble`) with the number of rows corresponding to the number of components of the `pgeometry` object given as input and two columns in the format `(geometry, md)`.

Examples

```
pcomp1 <- create_component("MULTIPOINT(1 2, 3 2)", 0.4)
pcomp2 <- create_component("POINT(2 1)", 0.3)
pcomp3 <- create_component("MULTIPOINT(5 1, 0 0)", 1)
ppoint <- create_pgeometry(list(pcomp1, pcomp2, pcomp3), "PLATEAUPOINT")

# Converting the pgeometry object into a tibble object
ppoint_tibble <- as_tibble(ppoint)
ppoint_tibble

# Converting it into data.frame
ppoint_df <- as.data.frame(ppoint)
ppoint_df
```

`component-class` *An S4 Class for representing a component of a spatial plateau object*

Description

An S4 Class for representing a component of a spatial plateau object

Details

A `component` object is composed of two attributes. The first one is a crisp spatial object and the second one is the membership degree in]0, 1] of this `component`.

Slots

`obj` An `sfg` object.

`md` The membership degree of the component.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

create_empty_pgeometry *Create an empty* pgeometry *object*

Description

create_empty_pgeometry() builds an empty pgeometry object of a specific type.

Usage

```
create_empty_pgeometry(type)
```

Arguments

type	A character value indicating the spatial plateau data type of the pgeometry object. It can be either "PLATEAUPOINT", "PLATEAULINE", "PLATEAUREGION", "PLATEAUCOMPOSITION" or "PLATEAUCOLLECTION".

Details

The create_empty_pgeometry() function creates a new pgeometry object with no components. To add new components to this object, you should use spa_add_component(). The components added to this object must be compatible with the type of the empty pgeometry object.

Value

An empty pgeometry object.

Examples

```
# Creating an empty plateau point object
empty_plateau_point <- create_empty_pgeometry("PLATEAUPOINT")
empty_plateau_point

# Creating an empty plateau line object
empty_plateau_line <- create_empty_pgeometry("PLATEAULINE")
empty_plateau_line

# Creating an empty plateau region object
empty_plateau_region <- create_empty_pgeometry("PLATEAUREGION")
empty_plateau_region

# Creating an empty plateau composition object
empty_plateau_composition <- create_empty_pgeometry("PLATEAUCOMPOSITION")
empty_plateau_composition

# Creating an empty plateau collection object
empty_plateau_collection <- create_empty_pgeometry("PLATEAUCOLLECTION")
empty_plateau_collection
```

create_pgeometry *Create a* pgeometry *object with components*

Description

create_pgeometry() creates a pgeometry object from a data.frame or tibble object, a list of components, or a list of spatial plateau objects.

Usage

```
create_pgeometry(x, type, is_valid = TRUE)
```

Arguments

x	A list of component objects, a list of pgeometry objects or a data.frame/tibble object. For PLATEAUPOINT, PLATEAULINE and PLATEAUREGION, the type of each component must be the same for all components.
type	A character value that indicates the type of the desired pgeometry object. It should be either "PLATEAUPOINT", "PLATEAULINE", "PLATEAUREGION", "PLATEAUCOMPOSITION", or "PLATEAUCOLLECTION". It must be compatible with the components given in x parameter.
is_valid	A Boolean value to check whether the user wants to validate the created spatial plateau object at the end. If is_valid = TRUE, it calls validObject() method.

Details

create_pgeometry() is a flexible function that creates a pgeometry object by using the values given in x. This object is built by using either a list of component objects, a list of pgeometry objects or a data.frame (or tibble) object. If a data.frame or tibble object is given as input, its columns must have the following format: (i) first column is an sfc object, and (ii) the second columns consists of the membership degree of each respective object of the sfc column.

By default, this function checks if the resulting spatial plateau object is valid. That is, it checks whether all constraints defined by the Spatial Plateau Algebra are satisfied. For instance, the components of a plateau point, plateau line, or plateau region must be adjacent or disjoint from each other and have to be unique membership degrees.

If you are sure that the component objects provided to this function satisfy all the constraints, then you can use is_valid = FALSE to improve the performance of this function.

Value

A pgeometry object.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of spatial plateau data types are explained in detail in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.
- Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

Examples

```
library(sf)

# Creating some components
pts <- rbind(c(0, 2), c(4, 2))
# Point components
pcp1 <- create_component(st_multipoint(pts), 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
# Line components
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)
lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)

# Creating spatial plateau objects from lists of components
pp <- create_pgeometry(list(pcp1, pcp2, pcp3), "PLATEAUPOINT")
pl <- create_pgeometry(list(lcp1, lcp3, lcp4), "PLATEAULINE")
pr <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")
pcm <- create_pgeometry(list(pcp1, pcp2, lcp1, lcp2, lcp3, rcp2), "PLATEAUCOMPOSITION")

# Creating a spatial plateau objects from a list of spatial plateau objects
pcl <- create_pgeometry(list(pp, pr, pcm), "PLATEAUCOLLECTION")

# Converting pp into a tibble
pp
tibble_pp <- as_tibble(pp)
tibble_pp

# Creating a spatial plateau point from the previous tibble
equivalent_pp <- create_pgeometry(tibble_pp, "PLATEAUPOINT")
equivalent_pp
```

`fsi_add_cs` *Add the consequent to an FSI model*

Description

`fsi_add_cs()` adds the consequent to a fuzzy spatial inference (FSI) model. It consists of a set of membership functions labeled with linguistic values.

Usage

```
fsi_add_cs(fsi, lvar, lvals, mfs, bounds)
```

Arguments

`fsi`	The FSI model instantiated with the `fsi_create()` function.
`lvar`	A character value that represents a linguistic variable of the consequent.
`lvals`	A character vector that contains linguistic values of the linguistic variable of the consequent.
`mfs`	A vector of membership functions (see examples below).
`bounds`	A numeric vector that represents the lower and upper bounds of the consequent domain.

Details

The `fsi_add_cs()` function adds the consequent to an FSI model. Each linguistic value defined in `lvals` has a corresponding membership function defined in `mfs`. Thus, these two parameters must have the same length. For instance, the first value of `lvals` defines the linguistic value of the first membership function in `mfs`. In `bounds`, the lower and upper values correspond to the first and second parameter, respectively.

Value

An FSI model populated with a consequent.

References

Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and formal definitions of FSI models are introduced in:

- Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
# Defining two different types of membership functions
trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

trim_mf <- function(a, b, c) {
  function(x) {
    pmax(pmin((x - a)/(b - a), (c - x)/(c - b), na.rm = TRUE), 0)
  }
}

# Creating the FSI model
fsi <- fsi_create("To visit or not to visit, that is the question",
                  default_conseq = trim_mf(10, 30, 60))

# Creating the vector with the linguistic values of the linguistic variable "visiting experience"
lvals_visiting_exp <- c("awful", "average", "great")

# Defining the membership function for each linguistic value
awful_mf <- trim_mf(0, 0, 20)
average_mf <- trim_mf(10, 30, 60)
great_mf <- trap_mf(40, 80, 100, 100)

# Adding the consequent to the FSI model
fsi <- fsi_add_cs(fsi, "visiting experience", lvals_visiting_exp,
                  c(awful_mf, average_mf, great_mf), c(0, 100))
```

`fsi_add_fsa` *Add an antecedent to an FSI model*

Description

`fsi_add_fsa()` adds a fuzzy spatial antecedent to a fuzzy spatial inference (FSI) model. A fuzzy spatial antecedent corresponds to a layer of fuzzy spatial objects (i.e., spatial plateau objects) that describe the different characteristics of the problem. The antecedent has a linguistic variable and its fuzzy spatial objects have linguistic values so that they are used in the IF part of fuzzy rules.

Usage

```
fsi_add_fsa(fsi, lvar, tbl)
```

Arguments

`fsi`	The FSI model instantiated with the `fsi_create()` function.
`lvar`	A character value that represents a linguistic variable of the antecedent.
`tbl`	A tibble with spatial plateau objects annotated with linguistic values of the linguistic variable specified by the above `lvar` parameter.

Details

The `fsi_add_fsa()` function adds a fuzzy spatial antecedent composed of a linguistic variable and its corresponding `pgeometry` objects annotated by linguistic values. The format of `tbl` is the same as the output of the function `spa_creator()`, allowing users to directly provide plateau region objects as input when designing FSI models.

Value

An FSI model populated with a fuzzy spatial antecedent.

References

Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and formal definitions of FSI models are introduced in:

- Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
library(tibble)

trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

trim_mf <- function(a, b, c) {
  function(x) {
    pmax(pmin((x - a)/(b - a), (c - x)/(c - b), na.rm = TRUE), 0)
  }
}

# Creating spatial plateau objects for the linguistic variable "accommodation price"
lvals_accom_price <- c("cut-rate", "affordable", "expensive")
cut_rate_mf <- trap_mf(0, 0, 10, 48)
affordable_mf <- trap_mf(10, 48, 80, 115)
expensive_mf <- trap_mf(80, 115, 10000, 10000)

# Example of point dataset
accom_price <- tibble(longitude = c(-74.0, -74.0, -74.0),
                      latitude = c(40.8, 40.75, 40.7),
                      price = c(150, 76, 60))

accom_price_layer <- spa_creator(accom_price, classes = lvals_accom_price,
                         mfs = c(cut_rate_mf, affordable_mf, expensive_mf))
```

```
# Creating the FSI model
fsi <- fsi_create("To visit or not to visit, that is the question",
                  default_conseq = trim_mf(10, 30, 60))

# Adding the fuzzy spatial antecedent to the FSI model
fsi <- fsi_add_fsa(fsi, "accommodation price", accom_price_layer)
```

`fsi_add_rules` *Add fuzzy rules to an FSI model*

Description

`fsi_add_rules()` adds the fuzzy rules set to a fuzzy spatial inference (FSI) model. A fuzzy rule must contain only linguistic variables and values included in the antecedent parts and consequent.

Usage

```
fsi_add_rules(fsi, rules, weights = rep(1, length(rules)))
```

Arguments

`fsi`	An FSI model instantiated with the `fsi_create()` function.
`rules`	A character vector containing the rules defined by the user. It follows a specific format, as detailed below.
`weights`	A numeric vector of weight values for each rule. Default values are 1.

Details

The `fsi_add_rules()` function adds fuzzy rules to an FSI model. The definition of a fuzzy rule is user-friendly since users can write it by using the *linguistic variables* and *linguistic values* previously defined and added to the FSI model (via `fsi_add_fsa()` and `fsi_add_cs()`).

A fuzzy rule has the format `IF A THEN B`, where `A` is called the antecedent and `B` the consequent of the rule such that `A` implies `B`. Further, `A` and `B` are statements that combine fuzzy propositions by using logical connectives like `AND` or `OR`. Each fuzzy proposition has the format `LVar is LVal` where `LVal` is a linguistic value in the scope of the linguistic variable `LVar`.

To avoid possible contradictions keep in mind the following items when specifying the rules:

- the order of the statements in the antecedent is not relevant.
- each linguistic variable has to appear at most one time in each fuzzy rule.
- only one kind of logical connective (i.e., `AND` or `OR`) must be used in the statements of the antecedent.

Value

An FSI model populated with a fuzzy rules set.

References

Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and formal definitions of FSI models are introduced in:

- Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
# Creating the FSI model from an example
fsi <- visitation()

# Creating a vector of fuzzy rules
## note that we make use of the linguistic variables and linguistic values previously defined
rules <- c(
 "IF accommodation review is reasonable AND
    food safety is low
  THEN visiting experience is awful",
 "IF accommodation price is expensive AND
    accommodation review is reasonable
  THEN visiting experience is awful",
 "IF accommodation price is affordable AND
    accommodation review is good AND
    food safety is medium
  THEN visiting experience is average",
 "IF accommodation price is affordable AND
    accommodation review is excellent AND
    food safety is high
  THEN visiting experience is great",
 "IF accommodation price is cut-rate AND
    accommodation review is excellent AND
    food safety is high
  THEN visiting experience is great")

# Adding these rules to the FSI model previously instantiated
fsi <- fsi_add_rules(fsi, rules)
```

`fsi_create` *Create an empty fuzzy spatial inference model*

Description

`fsi_create()` builds a fuzzy spatial inference (FSI) model without elements of the data source component (i.e., spatial plateau objects, fuzzy rules set, and fuzzy sets).

Usage

```
fsi_create(name, and_method = "min", or_method = "max",
           imp_method = "min", agg_method = "max",
           defuzz_method = "centroid", default_conseq = NULL)
```

Arguments

name	A character value that specifies the name of the FSI model.
and_method	A character value that defines the operator for the logical connective AND. Default value is "min".
or_method	A character value that defines the operator for the logical connective OR. Default value is "max".
imp_method	A character value that defines the implication operator. Default value is "min".
agg_method	A character value that defines the aggregation operator. Default value is "max".
defuzz_method	A character value that determines the defuzzification technique. Default value is the centroid technique.
default_conseq	A function object that corresponds to a membership function of the consequent.

Details

The fsi_create() function creates an empty FSI model and its default parameter values will implement a model using Mamdani's method.

The possible values for the parameters and_method and imp_method are: "min", "prod". The name of a user-defined t-norm function can also be informed here. The possible value for the parameters or_method and agg_method is: "max". The name of a user-defined t-conorm function can also be informed here. The possible values for the parameter defuzz_method are "centroid" (default value), "bisector", "mom", "som", and "lom". The parameter default_conseq defines the default behavior of the FSI model when there is no fuzzy rule with a degree of fulfillment greater than 0 returned by the FSI model.

After creating an empty FSI model, you have to call the functions fsi_add_fsa(), fsi_add_cs(), and fsi_add_rules() to fulfill the FSI model with the needed information before performing inferences.

Value

An empty named FSI model that is ready to be populated with data source component (i.e., spatial plateau objects, fuzzy rules set, and fuzzy sets).

References

Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and formal definitions of FSI models are introduced in:

- Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
trim_mf <- function(a, b, c) {
  function(x) {
    pmax(pmin((x - a)/(b - a), (c - x)/(c - b), na.rm = TRUE), 0)
  }
}

# Creating the FSI model
fsi <- fsi_create("To visit or not to visit, that is the question",
                  default_conseq = trim_mf(10, 30, 60))
```

`fsi_eval` *Evaluate a point inference query*

Description

`fsi_eval()` evaluates a point inference query. Considering an FSI model, it answers the following question: what is the inferred value for a given single point location?

Usage

```
fsi_eval(fsi, point, ...)
```

Arguments

`fsi`	An FSI model built with the `fsi_create()` and populated by `fsi_add_fsa()`, `fsi_add_cs()`, and `fsi_add_rules()`.
`point`	An `sfg` object of the type `POINT`.
`...`	<dynamic-dots> Informs the `fsi_eval` how the elements of the resulting fuzzy set should be discretized if the user does not want the default configuration (see below). Default values: `discret_by` is 0.5 and `discret_length` is `NULL`.

Details

The `fsi_eval()` function evaluates a point inference query by using an FSI model populated with its fuzzy spatial antecedent, consequent, and fuzzy rules set. This evaluation is based on the algorithm specified by the references below.

The default behavior of `fsi_eval()` in the parameter `...` is to consider a discrete interval of values with an increment of 0.5 between lower and upper values for the consequent domain (i.e., defined by `fsi_add_cs()` with the parameter `bounds`).

The user can modify the default behavior by using one of the following two ways:

- define a value for the parameter `discret_by` by changing the incremental value.
- define a desired length for the sequence of values domain of the consequent by using the parameter `discret_length`.

Value

A numeric value that belongs to the domain of the consequent of the FSI model and represents the result of a point inference query

References

Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and definitions on the evaluation of point inference queries are introduced in:

- Carniel, A. C.; Galdino, F.; Schneider, M. Evaluating Region Inference Methods by Using Fuzzy Spatial Inference Models. In Proceedings of the 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2022), pp. 1-8, 2022.
- Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
library(sf)

# Creating the FSI model from an example
fsi <- visitation()

# Creating a vector of fuzzy rules
## note that we make use of the linguistic variables and linguistic values previously defined
rules <- c(
 "IF accommodation review is reasonable AND
    food safety is low
  THEN visiting experience is awful",
 "IF accommodation price is expensive AND
    accommodation review is reasonable
  THEN visiting experience is awful",
 "IF accommodation price is affordable AND
    accommodation review is good AND
    food safety is medium
  THEN visiting experience is average",
 "IF accommodation price is affordable AND
    accommodation review is excellent AND
    food safety is high
  THEN visiting experience is great",
 "IF accommodation price is cut-rate AND
    accommodation review is excellent AND
    food safety is high
  THEN visiting experience is great")

# Adding these rules to the FSI model previously instantiated
fsi <- fsi_add_rules(fsi, rules)

# Evaluating a point inference query
```

```
fsi_eval(fsi, st_point(c(-74.0, 40.7)))
## Not run:
# Changing the default discretization
fsi_eval(fsi, st_point(c(-74.0, 40.7)), discret_by = 0.8)
fsi_eval(fsi, st_point(c(-74.0, 40.7)), discret_length = 200)

## End(Not run)
```

`fsi_qw_eval` *Evaluate region inference methods*

Description

`fsi_qw_eval()` implements two methods for evaluating region inference (RI) queries: (i) Linguistic value-based RI query, and (ii) Optimal RI query. The objective of these queries is to capture all points that intersect a search object (e.g., a query window) and whose inferred values fulfill some specific user requirements (e.g., the points with the maximum or minimum inferred values).

Usage

```
fsi_qw_eval(fsi, qw, approach = "discretization", ...)
```

Arguments

`fsi`	An FSI model built with the `fsi_create()` function and populated by the functions `fsi_add_fsa()`, `fsi_add_cs()`, and `fsi_add_rules()`.
`qw`	An `sfg` object representing the search object (e.g., a query window). It has to be an axis-aligned rectangle represented by a simple polygon object of 5 points (since the last coordinate pair closes the external ring of the rectangle).
`approach`	Defines which approach is employed to perform the region inference: `"discretization"` or `"pso"`. Default value is `"discretization"`.
`...`	<dynamic-dots> Different set of parameters required depending on the chosen approach (see more in details below).

Details

The `fsi_qw_eval()` function evaluates two types of RI queries:

- *Linguistic value-based RI query*, which answers the following type of question: what are the points that intersect a given search object and have inferred values that belong to a target linguistic value?
- *Optimal RI query*, which answers the following type of question: what are the points that intersect a given search object and have the maximum (or minimum) inferred values?

`fsi_qw_eval()` offers two different methods to answer these questions: (i) *discretization* method, and (ii) *optimization* method. Comparative analyses (see reference below) indicate that the discretization method should be employed to process linguistic value-based RI queries, while the optimization method is more adequate for processing optimal RI queries. The details below describe how to use these methods.

For the *discretization* method, two additional parameters are needed and must be informed by using the three-dots parameter `...`:

- `target_lval`: A character value that indicates the target linguistic value from the linguistic variable of the consequent.
- `k`: A numeric value that defines the number of points that will be captured from the query window and evaluated by `fsi_eval()`. Its square root has to an integer value. Alternatively, you can inform the number of columns and rows of the regular grid to be created on the query window by informing numeric values for `n_col` and `n_row`, respectively. Thus, these parameters can be given instead of the number `k`.

The *optimization* method employs the particle swarm optimization (PSO) algorithm. Thus, the parameter `approach = "pso"` must be set together with the following parameters:

- `what`: A character value that defines the user's goal, which can be either **maximize** or **minimize** inferred values. Thus, this parameter can be either `"max"` or `"min"`. The default value is `"max"`.
- `max_depth`: A numeric value that refers to the number of times that the query window is divided into subquadrants. The default value is equal to 2. For instance, a `max_depth = 2` means that the query window will be split into four subquadrants, where the PSO will be applied to each one as its search space.

In addition, the PSO algorithm has its own set of parameters:

- `maxit`: A numeric value that defines the maximum number of iterations. Default value is 50.
- `population`: A numeric value that defines the number of particles. Default value is 10.

Value

A tibble in the format `(points, inferred_values)`, where `points` is an `sfc` object and `inferred_values` are inferred values in the domain of the consequent of the FSI model.

References

Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and definitions on the evaluation of region inference methods are explained in:

- Carniel, A. C.; Galdino, F.; Schneider, M. Evaluating Region Inference Methods by Using Fuzzy Spatial Inference Models. In Proceedings of the 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2022), pp. 1-8, 2022.

Examples

```
library(sf)

# Creating the FSI model from an example
fsi <- visitation()

# Creating a vector of fuzzy rules
## note that we make use of the linguistic variables and linguistic values previously defined
rules <- c(
 "IF accommodation review is reasonable AND
    food safety is low
  THEN visiting experience is awful",
 "IF accommodation price is expensive AND
    accommodation review is reasonable
  THEN visiting experience is awful",
 "IF accommodation price is affordable AND
    accommodation review is good AND
    food safety is medium
  THEN visiting experience is average",
 "IF accommodation price is affordable AND
    accommodation review is excellent AND
    food safety is high
  THEN visiting experience is great",
 "IF accommodation price is cut-rate AND
    accommodation review is excellent AND
    food safety is high
  THEN visiting experience is great")

# Adding these rules to the FSI model previously instantiated
fsi <- fsi_add_rules(fsi, rules)

# Defining the query window
pts_qw1 <- rbind(c(-73.92, 40.68527), c(-73.75, 40.68527),
                 c(-73.75, 40.75), c(-73.92, 40.75), c(-73.92, 40.68527))
qw1 <- st_polygon(list(pts_qw1))

# Recall that our running example is based on a small set of point datasets
# This means that inferred values will likely be the same

## Not run:
# Example using the discretization method
fsi_qw_eval(fsi, qw1, approach = "discretization", target_lval = "great", k = 25)

# Example using the optimization method
fsi_qw_eval(fsi, qw1, approach = "pso", max_depth = 2)

## End(Not run)
```

`fsr_components` *Create a component*

Description

create_component() builds an object of class component. A component consists of a crisp spatial object (sfg object) labeled with a membership degree in]0, 1]. It is a flexible function since the crisp spatial object can be provided by using different formats.

Usage

```
create_component(obj, md, ...)

component_from_sfg(sfg, md)
```

Arguments

obj	A crisp spatial object in a specific format (see details below).
md	A numeric value indicating the membership degree of the component. It must be a value in]0, 1].
...	<dynamic-dots> Different parameters that are used to convert a crisp spatial object from a specific representation (see more in details below).
sfg	An sfg object. It should be either of type POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON or MULTIPOLYGON. Other types of spatial objects are not allowed.

Details

The create_component() function creates a component object. Internally, it is a pair of an sfg object and a membership degree in]0, 1].

obj can be either (see restrictions regarding its data type below):

- an sfg object.
- a character vector containing the WKT representation of a crisp spatial object.
- a structure of class "WKB" with the WKB or EWKB representation of a crisp spatial object. If the EWKB representation is used, then you have to provide the additional parameter EWKB = TRUE in ....
- a vector, list, or matrix containing coordinate pairs to be used when creating the sfg object. This means that it has a similar behavior to the family of functions st of the sf package (e.g., st_point(), st_multipoint(), etc.). Thus, you have to provide the additional parameter type in ..., which should be either "POINT", "LINE", or "REGION".

It is important to emphasize that the crisp spatial object must be a simple or complex point, line, or region (i.e., polygon) object. That is, it should be a POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON or MULTIPOLYGON object. If other types of crisp spatial objects are given, an error will be thrown.

The component_from_sfg() function is deprecated.

Value

A component object that can be added to a spatial plateau object (i.e., a pgeometry object).

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Examples

```
# first way: providing sfg objects
library(sf)

pts <- rbind(c(1, 2), c(3, 2))
comp1 <- create_component(st_multipoint(pts), 0.2)

lpts <- rbind(c(2, 2), c(3, 3))
comp2 <- create_component(st_linestring(lpts), 0.1)

matrix_obj <- matrix(c(1,1,8,1,8,8,1,8,1,1), ncol = 2, byrow = TRUE)
rpts <- list(matrix_obj)
comp3 <- create_component(st_polygon(rpts), 0.4)

# second way: providing WKT representations
comp4 <- create_component("POINT(10 35)", 0.5)
comp5 <- create_component("MULTILINESTRING((-29 -27, -36 -31, -45 -33), (-45 -33, -46 -32))", 0.9)
comp6 <- create_component("POLYGON((75 29, 77 29, 77 29, 75 29))", 1)

# third way: providing WKB representations
wkb = structure(list("0x0101000020e6100000000000000000000000000000000040"), class = "WKB")
comp7 <- create_component(wkb, 0.8, EWKB = TRUE)

# fourth way: providing coordinate pairs
coords1 = rbind(c(2,2), c(3,3))
coords2 = rbind(c(1,1), c(3,2))

comp8 <- create_component(coords1, 0.45, type = "LINE")
comp9 <- create_component(coords2, 0.32, type = "POINT")
```

---

fsr_diff_operators *Compute fuzzy difference operators*

Description

Fuzzy difference operations are set operations that generalize Boolean difference operations. This family of functions implements some operators that help us to define different fuzzy difference operations. These operators receive two numerical values in [0, 1] as input and calculates another numerical value in [0, 1] as output.

Usage

```
f_diff(x, y)
```

```
f_bound_diff(x, y)

f_symm_diff(x, y)

f_abs_diff(x, y)
```

Arguments

x	A numerical vector whose values are in [0, 1].
y	A numerical vector whose values are in [0, 1].

Details

These functions calculate the resulting membership degree of a fuzzy difference operator applied on two numerical values in the interval [0, 1]. The following fuzzy difference operators are available:

- `f_diff()`: The standard *fuzzy set difference* operator defined as the intersection of `x` and the complement of `y`, that is, `min(x, 1 - y)`.
- `f_bound_diff()`: The *fuzzy bounded difference* operator defined as `x` minus `y` with upper bound equal to 0, that is, `max(0, x - y)`.
- `f_symm_diff()`: The *fuzzy symmetric difference* operator defined as the union of the difference of `x` and `y` and the difference of `y` and `x`, that is, `max(f_diff(x, y), f_diff(y, x))`.
- `f_abs_diff()`: The *fuzzy absolute difference* operator defined as the absolute difference of `x` and `y`, that is, `abs(x - y)`.

The name of these functions can be used in the parameter `dtype` of the `spa_difference()` function.

Value

A numerical vector.

Examples

```
x <- c(0.1, 0.3, 0.6, 0.8)
y <- c(0.9, 0.7, 0.4, 0.2)

f_diff(x, y)
f_bound_diff(x, y)
f_symm_diff(x, y)
f_abs_diff(x, y)
```

`fsr_eval_modes` *Evaluate a membership degree*

Description

This family of functions implements evaluation modes that returns a Boolean value for a given degree in [0, 1] obtained from a membership function of a linguistic value.

Usage

```
soft_eval(degree)

strict_eval(degree)

alpha_eval(degree, alpha)

soft_alpha_eval(degree, alpha)
```

Arguments

`degree`	A numerical vector whose values are in [0, 1].
`alpha`	A single numeric value in [0, 1].

Details

These functions yield a Boolean value that indicates whether the membership degree matches an expected interpretation (according to the meaning of an evaluation mode). That is, the parameter `degree` is a value in [0, 1] and an evaluation mode "translates" the meaning of this degree of truth as a Boolean value.

There are some different ways to make this translation:

- `soft_eval()` returns `TRUE` if `degree` is greater than 0.
- `strict_eval()` returns `TRUE` if `degree` is equal to 1.
- `alpha_eval()` returns `TRUE` if `degree` is greater than or equal to another value (named `alpha`).
- `soft_alpha_eval()` returns `TRUE` if `degree` is greater than another value (named `alpha`).

These operators are employed to process the evaluation modes of fuzzy topological relationships (parameter `eval_mode`) that are processed as Boolean predicates.

Value

A Boolean vector.

Examples

```
x <- c(0, 0.1, 0.3, 0.6, 1, 0.8)

soft_eval(x)
strict_eval(x)
alpha_eval(x, 0.3)
soft_alpha_eval(x, 0.3)
```

`fsr_filter_operations` *Return a crisp spatial object formed by geometric parts of a* `pgeometry` *object*

Description

These functions yield a crisp spatial object (as an `sfg` object) formed by the geometric parts of the components of the `pgeometry` given as input that satisfy a filter condition based on their membership degrees.

Usage

```
spa_range(pgo, lvalue, rvalue, lside_closed = TRUE, rside_closed = TRUE)

spa_alpha_cut(pgo, alpha)

spa_strict_alpha_cut(pgo, alpha)
```

Arguments

`pgo`	A `pgeometry` object of any type.
`lvalue`	A numeric value denoting the left side of an interval in [0, 1].
`rvalue`	A numeric value denoting the right side of an interval in [0, 1].
`lside_closed`	A Boolean value indicating whether the left side is closed or not. The default value is `TRUE`.
`rside_closed`	A Boolean value indicating whether the right side is closed or not. The default value is `TRUE`.
`alpha`	A numeric value. For `spa_alpha_cut()`, it must be in [0, 1]. For `spa_strict_alpha_cut()`, it must be in]0, 1].

Details

Given a spatial plateau object as input, these functions return a crisp spatial object formed by the geometric parts of the components of the input that satisfy a filter condition based on their membership degrees. The filter condition of each function is detailed as follows:

- `spa_alpha_cut()` selects all components that have membership degrees greater than or equal to a given value in [0, 1] indicated by the parameter `alpha`.

- `spa_strict_alpha_cut()` picks a subset of components that have membership values greater than the parameter `alpha` (a value in]0, 1]).
- `spa_range()` generalizes these two operations and allows one to pick all components that have membership degrees belonging to a given open or closed interval. The parameters `lside_closed` and `rside_closed`, respectively, determine whether the left and right side (parameters `lvalue` and `rvalue`) of the interval is open (`FALSE`) or closed (`TRUE`). For example, to represent the right open interval [0.5, 0.8[, the following parameter values should be given: `lvalue = 0.5, rvalue = 0.8, lside_closed = TRUE, rside_closed = FALSE`.

Value

An `sfg` object that represents the geometric union of the components extracted after applying the specific filter condition.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Examples

```
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((1 2), (2 1), (3 2))", 1)
pcp5 <- create_component("MULTIPOINT((0 0.5), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3.5))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (4 2))", 0.4)

# Creating a plateau point object
ppoint <- create_pgeometry(list(pcp1, pcp2, pcp3, pcp4, pcp5), "PLATEAUPOINT")
ppoint

# Processing the alpha-cut, strict alpha-cut, and range
spa_alpha_cut(ppoint, 0.7)
spa_strict_alpha_cut(ppoint, 0.7)
spa_range(ppoint, 0.4, 0.8)
```

`fsr_geometric_operations`

Compute fuzzy geometric set operations

Description

The spatial plateau set operations *plateau intersection*, *plateau union*, and *plateau difference* implement the respective operations *fuzzy geometric intersection*, *fuzzy geometric union*, and *fuzzy geometric difference*.

Usage

```
spa_intersection(pgo1, pgo2, itype = "min", as_pcomposition = FALSE)

spa_union(pgo1, pgo2, utype = "max", as_pcomposition = FALSE)

spa_difference(pgo1, pgo2, dtype = "f_diff", as_pcomposition = FALSE)

spa_common_points(pline1, pline2, itype = "min")
```

Arguments

pgo1	A pgeometry object of any type.
pgo2	A pgeometry object of any type.
itype	A character value that indicates the name of a function implementing a t-norm. The default value is "min", which is the standard operator of the intersection.
as_pcomposition	A logical value; if TRUE, it returns a spatial plateau composition object.
utype	A character value that refers to a t-conorm. The default value is "max", which is the standard operator of the union.
dtype	A character value that indicates the name of a difference operator. The default value is "f_diff", which implements the standard fuzzy difference.
pline1	A pgeometry object of the type PLATEAULINE.
pline2	A pgeometry object of the type PLATEAULINE.

Details

They receive two pgeometry objects of the *any type* as input and yield another pgeometry object as output. The family of fuzzy geometric set operations consists of the following functions:

- spa_intersection() computes the geometric intersection of two spatial plateau objects. The membership degree of common points are calculated by using a t-norm operator given by the parameter itype. Currently, it can assume "min" (default) or "prod".
- spa_union() computes the geometric union of two spatial plateau objects. The membership degree of common points are calculated by using a t-conorm operator given by the parameter utype. Currently, it can assume "max" (default).
- spa_difference() computes the geometric difference of two spatial plateau objects. The membership degree of common points are calculated by using a difference operator given by the parameter dtype. Currently, it can assume "f_diff" (default fuzzy difference), "f_bound_diff" (fuzzy bounded difference), "f_symm_diff" (fuzzy symmetric difference), or "f_abs_diff" (fuzzy absolute difference).

Other t-norms, t-conorms, and difference operators can be implemented and given as values for the parameters itype, utype, and dtype, respectively. For this, the following steps should be performed:

1. Implement your function that accepts two numeric values in [0, 1] as inputs and yields another numeric value in [0, 1] as output. Recall that t-norms and t-conorms must have some specific properties according to the fuzzy set theory.

2. Use the name of your function as the character value of the corresponding parameter `itype`, `utype`, or `dtype`.

An example of operator is the source code of `f_bound_diff()`:

```
f_bound_diff <- function(x, y) { max(0, (x - y)) }
```

The `spa_common_points()` is deprecated. In the past, it computed the common points of two plateau line objects; now, you can use `spa_intersection()`.

Value

A `pgeometry` object that is the result of a fuzzy geometric set operation.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of spatial plateau set operations are explained in detail in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.
- Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

Examples

```
library(ggplot2)

# Point components
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)

pcp4 <- create_component("MULTIPOINT((2 2), (2 4), (3 2))", 1)
pcp5 <- create_component("MULTIPOINT((0 0), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (4 2))", 0.4)
# Line components
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)

lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
lcp5 <- create_component("LINESTRING(-1 1, 2 2)", 0.9)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)
```

```
# Creating plateau point objects
ppoint1 <- create_pgeometry(list(pcp1, pcp2, pcp3), "PLATEAUPOINT")
ppoint2 <- create_pgeometry(list(pcp4, pcp5, pcp6, pcp7), "PLATEAUPOINT")
# Creating plateau line objects
pline1 <- create_pgeometry(list(lcp1, lcp2, lcp3), "PLATEAULINE")
pline2 <- create_pgeometry(list(lcp4, lcp5), "PLATEAULINE")
# Creating a plateau region objects
pregion <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")

# Defining a wrapper to combine plots side by side, for convenience
combine_plots <- function(plot1, plot2, plot3) {
  # setting the same range of coordinates and removing the legend of plot1 and plot2
 plot1 <- plot1 + coord_sf(xlim = c(0, 4), ylim = c(0, 4)) + theme(legend.position = "none")
 plot2 <- plot2 + coord_sf(xlim = c(0, 4), ylim = c(0, 4)) + theme(legend.position = "none")
  plot3 <- plot3 + coord_sf(xlim = c(0, 4), ylim = c(0, 4))
  ggplot() +
    annotation_custom(ggplotGrob(plot1), xmin = 0, xmax = 0.5, ymin = 0.5, ymax = 1) +
    annotation_custom(ggplotGrob(plot2), xmin = 0.5, xmax = 1, ymin = 0.5, ymax = 1) +
    annotation_custom(ggplotGrob(plot3), xmin = 0, xmax = 1, ymin = 0, ymax = 0.5) +
    coord_cartesian(xlim = c(0, 1), ylim = c(0, 1)) +
    theme_void()
}

plot_ppoint1 <- plot(ppoint1) + ggtitle("Plateau point 1")
plot_ppoint2 <- plot(ppoint2) + ggtitle("Plateau point 2")
plot_pline1 <- plot(pline1) + ggtitle("Plateau line 1")
plot_pline2 <- plot(pline2) + ggtitle("Plateau line 2")
plot_pregion <- plot(pregion) + ggtitle("Plateau region")

# Computing the intersection
ppoints_intersec <- spa_intersection(ppoint1, ppoint2)
plot_inter <- plot(ppoints_intersec) + ggtitle("Intersection")
combine_plots(plot_ppoint1, plot_ppoint2, plot_inter)

## Not run:
# varying the t-norm
ppoints_intersec <- spa_intersection(ppoint1, ppoint2, itype = "prod")
plot_inter <- plot(ppoints_intersec) + ggtitle("Intersection (prod)")
combine_plots(plot_ppoint1, plot_ppoint2, plot_inter)

plines_intersec <- spa_intersection(pline1, pline2)
plot_inter <- plot(plines_intersec) + ggtitle("Intersection")
combine_plots(plot_pline1, plot_pline2, plot_inter)

pregion_pline_intersec <- spa_intersection(pline1, pregion)
plot_inter <- plot(pregion_pline_intersec) + ggtitle("Intersection")
combine_plots(plot_pline1, plot_pregion, plot_inter)

# Computing the union
ppoints_union <- spa_union(ppoint1, ppoint2)
plot_union <- plot(ppoints_union) + ggtitle("Union")
combine_plots(plot_ppoint1, plot_ppoint2, plot_union)
```

```
plines_union <- spa_union(pline1, pline2)
plot_union <- plot(plines_union) + ggtitle("Union")
combine_plots(plot_pline1, plot_pline2, plot_union)

pregion_pline_union <- spa_union(pline1, pregion)
plot_union <- plot(pregion_pline_union) + ggtitle("Union")
combine_plots(plot_pline1, plot_pregion, plot_union)

# Computing the difference
ppoints_diff <- spa_difference(ppoint1, ppoint2)
plot_diff <- plot(ppoints_diff) + ggtitle("Difference")
combine_plots(plot_ppoint1, plot_ppoint2, plot_diff)

plines_diff <- spa_difference(pline1, pline2)
plot_diff <- plot(plines_diff) + ggtitle("Difference")
combine_plots(plot_pline1, plot_pline2, plot_diff)

pregion_pline_diff <- spa_difference(pline1, pregion)
plot_diff <- plot(pregion_pline_diff) + ggtitle("Difference")
combine_plots(plot_pline1, plot_pregion, plot_diff)

## End(Not run)
```

`fsr_numerical_operations` *Compute fuzzy numerical operations*

Description

Fuzzy numerical operations are implemented by spatial plateau numerical operations, which extract geometric measurements from spatial plateau objects, such as the area of a plateau region object and the length of a plateau line object.

Usage

```
spa_avg_degree(pgo)

spa_ncomp(pgo)

spa_area(pgo)

spa_perimeter(pgo)

spa_length(pgo)
```

Arguments

`pgo`	A `pgeometry` object of the type `PLATEAULINE`, `PLATEAUCOMPOSITION`, or `PLATEAUCOLLECTION`. It throws a warning if a different type is given.

Details

These functions calculate numerical properties from spatial plateau objects (i.e., `pgeometry` objects). Some of them are *type-independent*. This means that the parameter can be a `pgeometry` object of any type. The type-independent functions are:

- `spa_avg_degree()` calculates the average membership degree of a spatial plateau object.
- `spa_ncomp()` returns the number of components of a spatial plateau object.

The remaining functions are *type-dependent*. This means that the parameter have to be of a specific type. The type-dependent functions are:

- `spa_area()` computes the area of a plateau region, plateau composition, or plateau collection object.
- `spa_perimeter()` computes the perimeter of a plateau region, plateau composition, or plateau collection.
- `spa_length()` computes the length of a plateau line, plateau composition, or plateau collection object.

For the aforementioned functions, if the input has the incorrect data type, it throws a warning message and returns 0.

Value

A numerical value.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of some spatial plateau numerical operations are introduced in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
# Point components
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((1 2), (2 1), (3 2))", 1)
pcp5 <- create_component("MULTIPOINT((0 0.5), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3.5))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (4 2))", 0.4)
# Line components
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)
```

```
lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
lcp5 <- create_component("LINESTRING(-1 1, 2 2)", 0.9)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)

# Creating spatial plateau objects
ppoint <- create_pgeometry(list(pcp1, pcp2, pcp3, pcp4, pcp5), "PLATEAUPOINT")
pline <- create_pgeometry(list(lcp1, lcp2, lcp3), "PLATEAULINE")
pregion <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")
pcomp <- create_pgeometry(list(pcp6, pcp7, lcp4, lcp5), "PLATEAUCOMPOSITION")
pcol <- create_pgeometry(list(ppoint, pline, pregion, pcomp), "PLATEAUCOLLECTION")

spa_avg_degree(ppoint)
spa_avg_degree(pline)
spa_avg_degree(pregion)
spa_avg_degree(pcomp)
spa_avg_degree(pcol)

spa_ncomp(ppoint)
spa_ncomp(pline)
spa_ncomp(pregion)
spa_ncomp(pcomp)
spa_ncomp(pcol)

spa_area(pregion)
spa_area(pcomp)
spa_area(pcol)

spa_perimeter(pregion)
spa_perimeter(pcomp)
spa_perimeter(pcol)

spa_length(pline)
spa_length(pcomp)
spa_length(pcol)
```

`fsr_topological_relationships`

Compute fuzzy topological relationships

Description

Fuzzy topological relationships are implemented by spatial plateau topological relationships. A fuzzy topological relationship expresses a particular relative position of two spatial plateau objects. Such a topological relationship determines the degree to which it holds for any two spatial plateau objects by a real value in the interval [0, 1].

Usage

```
spa_overlap(pgo1, pgo2, itype = "min", ret = "degree", ...)

spa_meet(pgo1, pgo2, itype = "min", ret = "degree", ...)

spa_disjoint(pgo1, pgo2, itype = "min", ret = "degree", ...)

spa_equal(pgo1, pgo2, utype = "max", ret = "degree", ...)

spa_inside(pgo1, pgo2, utype = "max", ret = "degree", ...)

spa_contains(pgo1, pgo2, utype = "max", ret = "degree", ...)
```

Arguments

pgo1	A pregion object.
pgo2	A pregion object.
itype	A character value that indicates the name of a function implementing a t-norm. The default value is "min", which is the standard operator of the intersection.
ret	A character value that indicates the return type of the fuzzy topological relationship. The default value is "degree" and other possible values are "list" and "bool".
...	<dynamic-dots> If ret = "bool", two additional parameters have to be informed, as described below.
utype	A character value that indicates the name of a function implementing a t-conorm. The default value is "max", which is the standard operator of the union.

Details

These functions implement the spatial plateau topological relationships between plateau region objects. The key idea of these relationships is to consider point subsets resulting from the combination of spatial plateau set operations and spatial plateau metric operations on spatial plateau objects for computing the resulting degree. The resulting degree can be also interpreted as a linguistic value.

The spatial plateau topological relationships are implemented by the following functions:

- spa_overlap() computes the overlapping degree of two plateau region objects. Since it uses the intersection operation, a t-norm operator can be given by the parameter itype. Currently, it can assume "min" (default) or "prod".
- spa_meet() computes the meeting degree of two plateau region objects. Similarly to spa_overlap, a t-norm operator can be given by the parameter itype.
- spa_disjoint() computes the disjointedness degree of two plateau region objects. Similarly to spa_overlap and spa_meet, a t-norm operator can be given by the parameter itype.
- spa_equal() computes how equal are two plateau region objects. Since it uses the union operation, a t-conorm operator can be given by the parameter utype. Currently, it can assume "max" (default).

- `spa_inside()` computes the containment degree of `pgo1` in `pgo2`. Similarly to `spa_equal()`, a t-conorm operator can be given by the parameter `utype`.
- `spa_contains()` changes the order of the operations `pgo1` ad `pgo2` when invoking `spa_inside()`.

The parameter `ret` determines the returning value of a fuzzy topological relationship. The default value is `"degree"` (default), which indicates that the function will return a value in [0, 1] that represents the degree of truth of a given topological relationship.

For the remainder possible values, the functions make use of a set of linguistic values that characterize the different situations of topological relationships. Each linguistic value has an associated membership function defined in the domain [0, 1]. The `fsr` package has a default set of linguistic values. You can use the function `spa_set_classification()` to change this set of linguistic values.

The remainder possible values for the parameter `ret` are:

- `ret = "list"` indicates that the function will return a named list containing the membership degree of the result of the predicate for each linguistic value (i.e., it employs the membership functions of the linguistic values).
- `ret = "bool"` indicates that the function will return a Boolean value indicating whether the degree returned by the topological relationship matches a given linguistic value according to an *evaluation mode*. The evaluation mode and the linguistic values have to be informed by using the parameters `eval_mode` and `lval`, respectively. The possible values for `eval_mode` are: `"soft_eval"`, `"strict_eval"`, `"alpha_eval"`, and `"soft_alpha_eval"`. They have different behavior in how computing the Boolean value from the membership function of a linguistic value. See the documentation of the functions `soft_eval()`, `strict_eval()`, `alpha_eval()`, and `soft_alpha_eval()` for more details. Note that the parameter `lval` only accept a character value belonging to the set of linguistic values that characterize the different situations of topological relationships.

Value

The returning value is determined by the parameter `ret`, as described above.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of spatial plateau topological relationships and fuzzy topological relationships are respectively introduced in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.
- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.

Examples

```
library(tibble)
library(sf)

set.seed(456)

# Generating some random points to create pgeometry objects by using spa_creator()
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 30),
             z = runif(10, min = 0, max = 50))

# Getting the convex hull on the points to clip plateau region objects during their constructions
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, base_poly = ch, fuzz_policy = "fcp", k = 2)

plot(pregions$pgeometry[[1]])
plot(pregions$pgeometry[[2]])

## Not run:
# Showing the different types of returning values
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]])
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "bool",
           eval_mode = "soft_eval", lval = "mostly")

## Examples for evaluating the other fuzzy topological relationships
spa_meet(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_disjoint(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_equal(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_inside(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_contains(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")

## End(Not run)
```

---

pcollection-class *An S4 Class for representing plateau collections (subclass of* pgeometry*)*

Description

An S4 Class for representing plateau collections (subclass of pgeometry)

Details

A pcollection object is composed of a list of spatial plateau objects and inherits the attribute supp from the class pgeometry (i.e., the support).

Slots

`supp` It is inherited from `pgeometry`.

`pgos` A list of spatial plateau objects.

References

Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

`pcollection_to_pcomposition`
Convert a plateau collection object into a plateau composition object

---

Description

`pcollection_to_pcomposition()` converts a plateau collection object into an equivalent plateau composition object.

Usage

```
pcollection_to_pcomposition(pcol)
```

Arguments

`pcol`	A `pcollection` object.

Details

The `pcollection_to_pcomposition()` function yields a `pcomposition` object that is equivalent to the `pcollection` object given as input by aggregating all spatial plateau objects by type.

Value

A `pcomposition` object.

References

Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

Examples

```
# Point components
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((10 10), (9 8), (7 7))", 1)
pcp5 <- create_component("MULTIPOINT((0 0), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (2 3))", 0.4)
# Line components
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)
lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
lcp5 <- create_component("LINESTRING(-1 1, 2 2)", 0.9)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)

# Creating plateau point objects
ppoint1 <- create_pgeometry(list(pcp1, pcp2, pcp3), "PLATEAUPOINT")
ppoint2 <- create_pgeometry(list(pcp4, pcp5), "PLATEAUPOINT")
ppoint3 <- create_pgeometry(list(pcp4, pcp5), "PLATEAUPOINT")
ppoint4 <- create_pgeometry(list(pcp6, pcp7), "PLATEAUPOINT")
# Creating plateau line objects
pline1 <- create_pgeometry(list(lcp1, lcp3), "PLATEAULINE")
pline2 <- create_pgeometry(list(lcp2, lcp4), "PLATEAULINE")
pline3 <- create_pgeometry(list(lcp5), "PLATEAULINE")
# Creating a plateau region objects
pregion <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")
# Creating a plateau composition object
pcomposition <- create_pgeometry(list(ppoint4, pline3), "PLATEAUCOMPOSITION")
# Creating plateau collection objects
pcol1 <- create_pgeometry(list(ppoint1, ppoint2, ppoint3, pline1), "PLATEAUCOLLECTION")
pcol2 <- create_pgeometry(list(pline2, pregion, pcomposition, pcol1), "PLATEAUCOLLECTION")
pcol2
plot(pcol2)
## Not run:
converted_pcomp <- pcollection_to_pcomposition(pcol2)
converted_pcomp
plot(converted_pcomp)

## End(Not run)
```

`pcomposition-class` *An S4 Class for representing plateau compositions (subclass of* `pgeometry`*)*

Description

An S4 Class for representing plateau compositions (subclass of `pgeometry`)

Details

A pcomposition object is composed of a ppoint object, pline object, pregion object and inherits the attribute supp from the class pgeometry (i.e., the support).

Slots

supp It is inherited from pgeometry.

ppoint A plateau point object.

pline A plateau line object.

pregion A plateau region object.

References

Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

---

pgeometry-class *An S4 Class for representing spatial plateau data types*

Description

An S4 Class for representing spatial plateau data types

Details

It is a superclass for representing spatial plateau data types. A pgeometry object stores an sfg object that represents the union of all crisp spatial objects of its components (i.e., the support).

Slots

supp An sfg object that stores the union of all spatial objects of the components of the spatial plateau object.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

pline-class *An S4 Class for representing plateau lines (subclass of* pgeometry*)*

Description

An S4 Class for representing plateau lines (subclass of pgeometry)

Details

A pline object is composed of a list of component objects and inherits the attribute supp from the class pgeometry (i.e., the support).

Slots

supp It is inherited from pgeometry.

component A list of components.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

plot *Graphically visualize* pgeometry *objects*

Description

The fsr_plot() function (and the S4 method plot()) plots a pgeometry object.

Usage

```
fsr_plot(pgo, base_poly = NULL, add_base_poly = TRUE, low = "white", high = "black",
         crs = NA, clip = FALSE, line_lwd = 1, region_lwd = 1, ...)

## S4 method for signature 'pgeometry,missing'
plot(x, y, ...)
```

Arguments

pgo	A pgeometry object of any type.
base_poly	An sfg object of the type POLYGON or MULTIPOLYGON. It can also be an sfc object with only one element of the type POLYGON or MULTIPOLYGON.
add_base_poly	A Boolean value that indicates whether base_poly will added to the visualization.

low	A character value that indicates the color for the lowest membership degree (i.e., 0). Default is "white".
high	A character value that indicates the color for the highest membership degree (i.e., 1). Default is "black".
crs	A numerical value that denotes the coordinate reference system (i.e., EPSG code) of the visualization. Default is NA.
clip	A Boolean value that indicates whether the boundaries of the components must be clipped by the sfg object base_poly (if it is not null).
line_lwd	A numeric value that specifies the line width of linear components.
region_lwd	A numeric value that specifies the line width of the boundaries of polygonal components.
...	<dynamic-dots> Optional parameters. They can be the same as the parameters of geom_sf() function from ggplot2.
x	A pgeometry object of any type.
y	Not applicable.

Details

The fsr_plot() function uses a ggplot2 package to built the resulting plot. It receives a pgeometry object as input (if it is empty, an empty graphics in obtained).

The low and high parameters are the colors for the minimum and maximum limits of the membership degrees. The default colors are "white" and "black", respectively. Other colors can be given in the same way that colors are informed to visualizations produced by the ggplot2 package.

It is possible to clip the geometric format of the components by using the parameter base_poly. The boundaries of this object can also be included in the visualization if the parameter add_base_poly is TRUE.

Since the returned value is a ggplot object, it can be further be customized (see examples below).

Value

A ggplot object.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Examples

```
library(sf)

pts <- rbind(c(0, 2), c(4, 2))
# Point components
pcp1 <- create_component(st_multipoint(pts), 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
# Line components
```

```
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)
lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)
# Creating spatial plateau objects
pp <- create_pgeometry(list(pcp1, pcp2, pcp3), "PLATEAUPOINT")
pl <- create_pgeometry(list(lcp1, lcp3, lcp4), "PLATEAULINE")
pr <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")
pcm <- create_pgeometry(list(pcp1, pcp2, lcp1, lcp2, lcp3, rcp2), "PLATEAUCOMPOSITION")
pcl <- create_pgeometry(list(pp, pr, pcm), "PLATEAUCOLLECTION")

# Displaying their textual representations
pp
pl
pr
pcm
pcl

# Plotting them
plot(pp)
plot(pl)
plot(pr)
plot(pcm)
plot(pcl)
## Not run:
# Custom colors
fsr_plot(pr, low = "green", high = "blue")

# Changing the line width of line components
fsr_plot(pl, line_lwd = 2)

# Changing the line width of boundary lines of region components
fsr_plot(pr, region_lwd = 2)

# Changing the line width of boundary lines of region components and its color
fsr_plot(pr, region_lwd = 2, color = "blue")

# You can customize the whole visualization using ggplot
library(ggplot2)

fsr_plot(pp, size = 5) +
  theme(legend.position = "none") +
  theme(text=element_text(size=20, family = "serif", color = "black"),
        axis.text=element_text(color="black")) +
  scale_x_continuous(breaks = c(0, 1, 2, 3, 4)) +
  scale_y_continuous(breaks = c(0, 1, 2, 3, 4))

## End(Not run)
```

ppoint-class *An S4 Class for representing plateau points (subclass of* pgeometry*)*

Description

An S4 Class for representing plateau points (subclass of pgeometry)

Details

A ppoint object is composed of a list of component objects and inherits the attribute supp from the class pgeometry (i.e., the support).

Slots

supp It is inherited from pgeometry.

component A list of components.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

pregion-class *An S4 Class for representing plateau regions (subclass of* pgeometry*)*

Description

An S4 Class for representing plateau regions (subclass of pgeometry)

Details

A pregion object is composed of a list of component objects and inherits the attribute supp from the class pgeometry (i.e., the support).

Slots

supp It is inherited from pgeometry.

component A list of components.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

PWKT *Return PWKT representation of a spatial plateau object*

Description

These functions give the Plateau Well-Known Text (PWKT) representation of a pgeometry object.

Usage

```
spa_pwkt(pgo)

## S3 method for class 'pgeometry'
format(x, ..., width = 30)

## S4 method for signature 'pgeometry'
show(object)

## S4 method for signature 'pgeometry'
as.character(x, ...)
```

Arguments

pgo	A pgeometry object of any type.
x	A pgeometry object of any type.
...	<dynamic-dots> Unused.
width	An integer value that indicates the number of characters to be printed. If it is 0 NULL or NA, then it will print everything.
object	A pgeometry object of any type.

Details

These functions return the textual representation of a pgeometry object, which combines the Well-Known Text (WKT) representation for crisp vector geometry objects and the formal definitions of spatial plateau data types. (i.e. PLATEAUPOINT, PLATEAULINE, PLATEAUREGION, PLATEAUCOMPOSITION, and PLATEAUCOLLECTION).

Value

A character object (i.e., string) with the textual representation of a given pgeometry object.

References

The formal definition of PWKT is given in:

- Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of spatial plateau data types are explained in detail in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.
- Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

Examples

```
pcomp1 <- create_component("MULTIPOINT(1 2, 3 2)", 0.4)
pcomp2 <- create_component("POINT(2 1)", 0.3)
ppoint <- create_pgeometry(list(pcomp1, pcomp2), "PLATEAUPOINT")

# using spa_pwkt()
spa_pwkt(ppoint)
# using show() to display the content of ppoint
ppoint
# using format with width = 30 (default value)
format(ppoint)

lcomp1 <- create_component("LINESTRING(1 2, 3 3, 3 4)", 1)
lcomp2 <- create_component("LINESTRING(0 0, 5 5)", 0.5)
pline <- create_pgeometry(list(lcomp1, lcomp2), "PLATEAULINE")

spa_pwkt(pline)

rcomp1 <- create_component("POLYGON((40 40, 20 48, 48 35, 40 40))", 0.8)
rcomp2 <- create_component("POLYGON((10 0, 40 18, 10 20, 5 18, 10 0))", 0.2)
pregion <- create_pgeometry(list(rcomp1, rcomp2), "PLATEAUREGION")

spa_pwkt(pregion)

pcomposition <- create_pgeometry(list(ppoint, pline, pregion), "PLATEAUCOMPOSITION")

spa_pwkt(pcomposition)

pcomp3 <- create_component("POINT(10 15)", 0.3)
ppoint2 <- create_pgeometry(list(pcomp3), "PLATEAUPOINT")

pcollection <- create_pgeometry(list(pcomposition, ppoint2), "PLATEAUCOLLECTION")

spa_pwkt(pcollection)
```

---

spa_add_component *Add components to a* pgeometry *object*

Description

spa_add_component() inserts components into a spatial plateau object (i.e., pgeometry object).

Usage

```
spa_add_component(pgo, components, is_valid = FALSE)
```

Arguments

pgo	A pgeometry object of any type.
components	A component object or a list of component objects.
is_valid	A Boolean value to check if the user wants to validate the updated spatial plateau object at the end. If is_valid = TRUE, it calls validObject() method.

Details

This function implements the $\odot$ operator defined by Spatial Plateau Algebra. The goal of this function is to insert a component or a list of components into a pgeometry object. The crisp spatial object of the component must be compatible with the type of the plateau spatial object. For instance, a pregion object accepts only components containing polygons (e.g., POLYGON or MULTIPOLYGON). In the case of pcomposition object any type of component is compatible to be added. For instance, a point component is added to the plateau point sub-object of the plateau composition object. On the other hand, as a pcollection object can have multiple spatial objects of the same type, this function is not applicable to it.

The insertion is based on the membership degree of the component. Thus, it preserves the properties of a spatial plateau object. However, spa_add_component() assumes that the geometric format of the component is valid (i.e., it does not overlap with existing components).

Value

A pgeometry object containing the component objects.

References

The formal definition of the $\odot$ operator is described in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
comp1 <- create_component("MULTIPOINT(1 1, 2 2)", 0.2)
comp2 <- create_component("POINT(1 5)", 0.8)

# appending these components into an empty pgeometry object
pp <- create_empty_pgeometry("PLATEAUPOINT")
pp <- spa_add_component(pp, list(comp1, comp2))
pp

# inserting components with existing membership degrees are merged
comp3 <- create_component("MULTIPOINT(0 0, 4 4)", 0.2)
pp <- spa_add_component(pp, comp3)
```

```
pp

comp4 <- create_component("MULTIPOINT(0 1, 3 4)", 1)
pc <- create_pgeometry(list(comp4), "PLATEAUCOMPOSITION")
pc

# appending these components into pc
comp5 <- create_component("LINESTRING(-1 1, 2 2)", 0.9)
comp6 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
pc <- spa_add_component(pc, list(comp5, comp6))
pc
```

---

spa_boundary *Capture the fuzzy boundary of a spatial plateau object*

Description

spa_boundary() yields the fuzzy boundary of a homogeneous spatial plateau object.

Usage

```
spa_boundary(pgo)
```

Arguments

pgo	A pgeometry object of type ppoint, pline, or pregion.

Details

The spa_boundary() function employs the definition of *fuzzy boundary* in the context of Spatial Plateau Algebra. The *fuzzy boundary* of a fuzzy spatial object has a heterogeneous nature. For instance, the fuzzy boundary of a plateau region object consists of two parts:

- a plateau line object that corresponds to the boundary of the core of A.
- a plateau region object that comprises all points of A with a membership degree greater than 0 and less than 1.

This means that spa_boundary() returns a pcomposition object.

Value

A pcomposition object that represents a fuzzy boundary of the pgeometry object given as input.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Concepts and formal definitions of fuzzy boundary are introduced in:

- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.
- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(sf)
library(ggplot2)

# defining two different types of membership functions
trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

set.seed(7)
tbl = tibble(x = runif(20, min = 0, max = 30),
             y = runif(20, min = 0, max = 50),
             z = runif(20, min = 0, max = 100))
classes <- c("cold", "hot")
cold_mf <- trap_mf(0, 10, 20, 35)
hot_mf <- trap_mf(20, 50, 100, 100)

# Getting the convex hull on the points to clip plateau region objects during their constructions
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

# Using the standard fuzzification policy based on fuzzy sets
pregions <- spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf), base_poly = ch)
## Not run:
pregions
plot(pregions$pgeometry[[1]]) + ggtitle("Cold")
plot(pregions$pgeometry[[2]]) + ggtitle("Hot")

## End(Not run)
# capturing and showing the boundary of each pgeometry object previously created
boundary_cold <- spa_boundary(pregions$pgeometry[[1]])
boundary_hot <- spa_boundary(pregions$pgeometry[[2]])
## Not run:
plot(boundary_cold) + ggtitle("Boundary (Cold)")
plot(boundary_hot) + ggtitle("Boundary (Hot)")
```

```
## End(Not run)
```

---

spa_boundary_pregion *Capture the fuzzy boundary of a plateau region object*

Description

spa_boundary_pregion() yields a specific part of the fuzzy boundary of a plateau region object. This function is deprecated; use spa_boundary().

Usage

```
spa_boundary_pregion(pregion, bound_part = "region")
```

Arguments

pregion	A pregion object. It throws an error if a different type is given.
bound_part	A character value that indicates the part of the fuzzy boundary to be returned. It can be "region" or "line". See below for more details.

Details

The spa_boundary_pregion() function employs the definition of *fuzzy boundary* of a fuzzy region object in the context of Spatial Plateau Algebra. The *fuzzy boundary* of a fuzzy region object A has a heterogeneous nature since it consists of two parts:

- a fuzzy line object that corresponds to the boundary of the core of A.
- a fuzzy region object that comprises all points of A with a membership degree greater than 0 and less than 1.

This means that spa_boundary_pregion() can yield one specific part of the fuzzy boundary of a plateau region object. If boundary = "line", then the function returns the boundary plateau line of pregion (i.e., returns a pline object). Else if boundary = "region" (the default value), then the function returns the boundary plateau region of pregion (i.e., returns a pregion object).

This function is deprecated; use spa_boundary().

Value

A pgeometry object that represents a specific part of the fuzzy boundary of pgeometry object given as input.

References

Concepts of fuzzy boundary of plateau region objects are introduced in:

- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.
- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
## Not run:
library(tibble)
library(sf)
library(ggplot2)

# defining two different types of membership functions
trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

trim_mf <- function(a, b, c) {
  function(x) {
    pmax(pmin((x - a)/(b - a), (c - x)/(c - b), na.rm = TRUE), 0)
  }
}

set.seed(7)
tbl = tibble(x = runif(10, min = 0, max = 30),
             y = runif(10, min = 0, max = 50),
             z = runif(10, min = 0, max = 100))
classes <- c("cold", "hot")
cold_mf <- trap_mf(0, 10, 20, 35)
hot_mf <- trim_mf(35, 50, 100)

# Getting the convex hull on the points to clip plateau region objects during their constructions
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

# Using the standard fuzzification policy based on fuzzy sets
pregions <- spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf), base_poly = ch)
plot(pregions$pgeometry[[1]]) + ggtitle("Cold")
plot(pregions$pgeometry[[2]]) + ggtitle("Hot")

# these functions are now deprecated, use `spa_boundary()`

# capturing and showing the boundary plateau line of each pgeometry object previously created
(spa_boundary_pregion(pregions$pgeometry[[1]], bound_part = "line"))
(spa_boundary_pregion(pregions$pgeometry[[2]], bound_part = "line"))
```

```
# this part of the boundary is empty because there is no core!
# capturing and showing the boundary plateau region (this is the default behavior)
(spa_boundary_pregion(pregions$pgeometry[[1]]))
(spa_boundary_pregion(pregions$pgeometry[[2]]))

## End(Not run)
```

`spa_contour` *Capture the frontier of a plateau region object*

Description

`spa_contour()` extracts the frontier (i.e., linear boundary) of a plateau region object by maintaining its membership degrees.

Usage

```
spa_contour(pregion)
```

Arguments

`pregion`	A `pregion` object. It throws an error if a different type is given.

Details

The `spa_contour()` function implements the definition of *fuzzy frontier* of a fuzzy region object in the context of Spatial Plateau Algebra. The *fuzzy frontier* of a fuzzy region object `A` collects all single points of `A`, preserving its membership degrees, that are not in the interior of its support.

Note that fuzzy frontier is different from fuzzy boundary (see `spa_boundary()`).

Value

A `pline` object that represents the contour (i.e. frontier) of a plateau region object given as input.

References

- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.
- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(sf)
library(ggplot2)

# defining two different types of membership functions
trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

set.seed(7)
tbl = tibble(x = runif(20, min = 0, max = 30),
             y = runif(20, min = 0, max = 50),
             z = runif(20, min = 0, max = 100))
classes <- c("cold", "hot")
cold_mf <- trap_mf(0, 10, 20, 35)
hot_mf <- trap_mf(20, 50, 100, 100)

# Getting the convex hull on the points to clip plateau region objects during their constructions
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

# Using the standard fuzzification policy based on fuzzy sets
pregions <- spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf), base_poly = ch)
pregions
## Not run:
plot(pregions$pgeometry[[1]]) + ggtitle("Cold")
plot(pregions$pgeometry[[2]]) + ggtitle("Hot")

## End(Not run)
# capturing and showing the frontier of each pgeometry object previously created
cold_contour <- spa_contour(pregions$pgeometry[[1]])
hot_contour <- spa_contour(pregions$pgeometry[[2]])
## Not run:
plot(cold_contour) + ggtitle("Frontier (Cold)")
plot(hot_contour) + ggtitle("Frontier (Hot)")

## End(Not run)
```

---

spa_core *Get the core of a* pgeometry *object*

Description

spa_core() yields a crisp spatial object (as an sfg object) that corresponds to the core of a pgeometry object given as input.

Usage

```
spa_core(pgo)
```

Arguments

pgo	A pgeometry object of any type.

Details

The spa_core() function employs the classical definition of *core* from the fuzzy set theory in the context of Spatial Plateau Algebra. The *core* only comprises the points with membership degree equal to 1. Hence, this operation returns the sfg object that represents the component labeled with membership degree equal to 1 of the pgeometry object given as input. If the pgeometry object has no core, then an empty sfg object is returned.

Value

An sfg object that represents the core of pgo. It can be an empty object, if pgo does not have a component with membership degree 1.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of Spatial Plateau Algebra are introduced in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((1 2), (2 1), (3 2))", 1)
pcp5 <- create_component("MULTIPOINT((0 0.5), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3.5))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (4 2))", 0.4)

# Creating a plateau point object
ppoint <- create_pgeometry(list(pcp1, pcp2, pcp3, pcp4, pcp5), "PLATEAUPOINT")
ppoint

# Getting its core
spa_core(ppoint)

# Getting the core of an empty pgeometry
spa_core(create_empty_pgeometry("PLATEAUREGION"))
```

spa_creator *Build* pgeometry *objects from a point dataset*

Description

spa_creator() builds a set of spatial plateau objects from a given point dataset assigned with domain-specific numerical values.

Usage

```
spa_creator(tbl, fuzz_policy = "fsp", const_policy = "voronoi", ...)
```

Arguments

tbl	A data.frame or tibble object with three columns: (x, y, z).
fuzz_policy	The fuzzification policy to be employed by the algorithm. See details below.
const_policy	The construction policy to be used by the algorithm. See details below.
...	<dynamic-dots> Parameters for the chosen policies. See details below.

Details

The spa_creator() function implements a two-stage construction method that takes as input a point dataset and produces a set of spatial plateau objects as output.

The input tbl is a point dataset (data.frame or tibble object) where each point represents the location of a phenomenon treated by the application. Further, each point is annotated with numerical data that describe its meaning in the application. Therefore, tbl must have three columns: (x, y, z). The columns x, y are the coordinate pairs, and z is the column containing domain-specific numeric values.

The parameter fuzz_policy refers to the method used by the **fuzzification stage**. This stage aims to assign membership degrees to each point of the dataset. It accepts two possible values: "fsp" (default) or "fcp".

"fsp" stands for *fuzzy set policy* and requires two parameters that should be informed in ...:

- classes: A character vector containing the name of classes.
- mfs: A vector of membership functions. Each membership function i represents the class i, where i in length(classes). See the provided examples for more information on how to build membership functions.

"fcp" stands for *fuzzy clustering policy* and requires the e1071 package. Its possible parameters informed in ... are:

- k: A numeric value that refers to the number of groups to be created.
- method: A fuzzy clustering method of the package e1071, which can be either "cmeans" (default) or "cshell".
- use_coords: A Boolean value to indicate whether the columns (x, y) should be used in the clustering algorithm (default is FALSE).

- `iter`: A numeric indicating the number of maximum iterations of the clustering algorithm (default is 100).

An optional and common parameter for both fuzzification stages is `"digits"`. This is an integer value that indicates the number of decimal digits of the membership degrees calculated by the fuzzification stage. That is, it is used to **round** membership degrees to the specified number of decimal places. Be careful with this optional parameter! If you specify a low value for `"digits"`, some membership degrees could be rounded to 0 and thus, some components would not be created.

The parameter `const_policy` refers to the method used by the **construction stage**. This stage aims to create polygons from the labeled point dataset and use them to build spatial plateau objects. It accepts three possible values: `"voronoi"` (default), `"delaunay"`, or `"convex_hull"`.

`"voronoi"` stands for *Voronoi diagram policy* and has two optional parameter that can be provided in `...`:

- `base_poly`: An `sfg` object that will be used to clip the generated polygons. If this parameter is not provided, the Voronoi is created by using a bounding box (standard behavior of the package `sf`).
- `d_tolerance`: It refers to the parameter `dTolerance` employed by the function `st_voronoi()` of the package `sf`.

`"delaunay"` stands for *Delaunay triangulation policy*, which accepts the following parameters in `...`:

- `base_poly`: An `sfg` object that will be used to clip the generated triangles.
- `tnorm`: A t-norm used to calculate the membership degree of the triangle. It should be the name of a vectorized function. Possible values are `"min"` (default) and `"prod"`. Note that it is possible to use your own t-norms. A t-norm should has the following signature: `FUN(x)` where *x* is a numeric vector. Such a function should return a single numeric value.
- `d_tolerance`: It refers to the parameter `dTolerance` employed by the function `st_triangulate()` of the package `sf`.

`"convex_hull"` stands for *Convex hull policy*, which accepts the following parameters in `...`:

- `degrees`: A numeric vector containing the membership degrees that will be used to create the components. The default vector is defined by `seq(0.05, 1, by = 0.05)`.
- `d`: A numeric value representing the tolerance distance to compute the membership degree between the elements of `m` and the membership degrees of the points. The default is `0.05`.
- `base_poly`: An `sfg` object that will be used to clip the generated polygons.

Value

A tibble in the format `(class, pgeometry)`, where `class` is a character column and `pgeometry` is a list of `pgeometry` objects. This means that a spatial plateau object is created for representing a specific class of the point dataset.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of the two-stage construction method is introduced in:

- Carniel, A. C.; Schneider, M. A Systematic Approach to Creating Fuzzy Region Objects from Real Spatial Data Sets. In Proceedings of the 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2019), pp. 1-6, 2019.

Examples

```
library(tibble)
library(sf)
library(ggplot2)

# Defining two different types of membership functions
trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

trim_mf <- function(a, b, c) {
  function(x) {
    pmax(pmin((x - a)/(b - a), (c - x)/(c - b), na.rm = TRUE), 0)
  }
}

set.seed(7)
tbl = tibble(x = runif(10, min = 0, max = 30),
             y = runif(10, min = 0, max = 50),
             z = runif(10, min = 0, max = 100))
classes <- c("cold", "hot")
cold_mf <- trap_mf(0, 10, 20, 35)
hot_mf <- trim_mf(35, 50, 100)

# Using the standard fuzzification policy based on fuzzy sets
res <- spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf))
## Not run:
res
plot(res$pgeometry[[1]]) + ggtitle("Cold")
plot(res$pgeometry[[2]]) + ggtitle("Hot")

# Getting the convex hull on the points to clip plateau region objects during their constructions
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))
res <- spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf), base_poly = ch)
plot(res$pgeometry[[1]]) + ggtitle("Cold (with clipped boundaries)")
plot(res$pgeometry[[2]]) + ggtitle("Hot (with clipped boundaries)")

# Using the fuzzification policy based on fuzzy clustering
spa_creator(tbl, fuzz_policy = "fcp", k = 4)
```

```
spa_creator(tbl, fuzz_policy = "fcp", k = 4, digits = 2)

# Varying the construction policy
spa_creator(tbl, fuzz_policy = "fcp", k = 3, const_policy = "delaunay")

spa_creator(tbl, fuzz_policy = "fcp", const_policy = "delaunay", k = 3, tnorm = "prod")

spa_creator(tbl, fuzz_policy = "fcp", k = 2, digits = 2,
            degrees = seq(0.1, 1, by = 0.1), d = 0.05, const_policy = "convex_hull")

spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf), const_policy = "delaunay")

spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf),
            digits = 2, const_policy = "convex_hull")

## End(Not run)
```

---

spa_eval *Evaluate the membership degree of a point in a* pgeometry *object*

Description

spa_eval() evaluates the membership degree of a given point in a spatial plateau object of any type. It returns a value in [0, 1] that indicates to which extent the point belongs to the pgeometry object.

Usage

```
spa_eval(pgo, point)
```

Arguments

pgo	A pgeometry object of any type.
point	An sfg object of the type POINT.

Details

The spa_eval() returns the membership degree of a simple point object (i.e., sfg object) in a given spatial plateau object (i.e., pgeometry object). This evaluation depends on the following basic cases:

- if the simple point object belongs to the interior or boundary of *one* component of the spatial plateau object, it returns the membership degree of that component.
- if the simple point object intersects more components (e.g., boundaries of region components, or different line components), it returns the maximum membership degree of all intersected components.
- if the simple point object is disjoint to the support of the spatial plateau object, it returns 0.

Value

A numeric value between 0 and 1 that indicates the membership degree of a point (i.e., `sfg` object) in a spatial plateau object (i.e., `pgeometry` object).

References

Formal definitions of this function are described in:

- Carniel, A. C.; Galdino, F.; Schneider, M. Evaluating Region Inference Methods by Using Fuzzy Spatial Inference Models. In Proceedings of the 2022 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2022), pp. 1-8, 2022.
- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)

# Point components
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((1 2), (2 1), (3 2))", 1)
pcp5 <- create_component("MULTIPOINT((0 0.5), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3.5))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (4 2))", 0.4)
# Line components
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)
lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
lcp5 <- create_component("LINESTRING(-1 1, 2 2)", 0.9)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)

# Creating spatial plateau objects
ppoint <- create_pgeometry(list(pcp1, pcp2, pcp3, pcp4, pcp5), "PLATEAUPOINT")
pline <- create_pgeometry(list(lcp1, lcp2, lcp3), "PLATEAULINE")
pregion <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")
pcomp <- create_pgeometry(list(pcp6, pcp7, lcp4, lcp5), "PLATEAUCOMPOSITION")
pcol <- create_pgeometry(list(ppoint, pline, pregion, pcomp), "PLATEAUCOLLECTION")

point <- st_point(c(0, 0))

spa_eval(ppoint, point)
spa_eval(pline, point)
spa_eval(pregion, point)
spa_eval(pcomp, point)
spa_eval(pcol, point)
```

spa_exact_equal *Check two spatial plateau objects for exact equality*

Description

spa_exact_equal() checks whether two spatial plateau objects are exactly equal.

Usage

```
spa_exact_equal(pgo1, pgo2)
```

Arguments

pgo1	A pgeometry object that is either a plateau point, plateau line, or plateau region object.
pgo2	A pgeometry object that is either a plateau point, plateau line, or plateau region object.

Details

spa_exact_equal() is a Boolean function that checks *fuzzy equality* in the spatial plateau context. Two pgeometry objects are exactly equal if their components are equal. Two components are equal if they have the same membership degree and they are (spatially) equal (i.e., their sfg objects have the same geometric format - this means that the order of the points can be different).

Value

A Boolean value that indicates if two pgeometry objects are exactly equal.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
pcp1 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp2 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp3 <- create_component("MULTIPOINT((10 10), (9 8), (7 7))", 1)
pcp4 <- create_component("MULTIPOINT((0 0), (2 3))", 0.7)

ppoint1 <- create_pgeometry(list(pcp1, pcp2), "PLATEAUPOINT")
ppoint2 <- create_pgeometry(list(pcp3, pcp4), "PLATEAUPOINT")

spa_exact_equal(ppoint1, ppoint2)

spa_exact_equal(ppoint1, ppoint1)
```

spa_exact_inside *Check two spatial plateau objects for exact containment*

Description

`spa_exact_inside()` checks whether a `pgeometry` object is completely inside of another `pgeometry` object.

Usage

```
spa_exact_inside(pgo1, pgo2)
```

Arguments

`pgo1`	A `pgeometry` object that is either a plateau point, plateau line, or plateau region object.
`pgo2`	A `pgeometry` object that is either a plateau point, plateau line, or plateau region object.

Details

`spa_exact_inside()` is a Boolean function that checks *fuzzy containment* in the spatial plateau context. This Boolean function checks whether the components of `pgo1` are contained in the components of `pgo2` by considering their membership degrees and geographic positions. That is, it follows the classical definition of fuzzy containment of the fuzzy set theory.

In other words, this function checks if the (standard) intersection of `pgo1` and `pgo2` is exactly equal to `pgo1`. The other of operands affects the result.

Value

A Boolean value that indicates if a `pgeometry` is completely and certainly inside `pgo2`.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
pcp1 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp2 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp3 <- create_component("POINT(2 2)", 0.2)
pcp4 <- create_component("MULTIPOINT((1 1), (3 3))", 0.7)

ppoint1 <- create_pgeometry(list(pcp1, pcp2), "PLATEAUPOINT")
ppoint2 <- create_pgeometry(list(pcp3, pcp4), "PLATEAUPOINT")
```

```
# is ppoint2 completely and certainly inside ppoint1?
spa_exact_inside(ppoint2, ppoint1)

# The order of operands after the result
# ppoint1 is not inside ppoint2 since it has different points
spa_exact_inside(ppoint1, ppoint2)
```

`spa_flatten` *Flatten a plateau collection object*

Description

`spa_flatten()` gathers all the objects of a plateau collection object and reorganizes them into a single flattened spatial plateau object containing a quadruple (`PLATEAUPOINT`, `PLATEAULINE`, `PLATEAUREGION`, `PLATEAUCOMPOSITION`) that preserves the identity of sub-objects.

Usage

```
spa_flatten(pcol)
```

Arguments

`pcol`	A `pcollection` object.

Details

The `spa_flatten()` function yields a single flattened spatial plateau object, aggregating all spatial plateau objects by their types. In the case of a two-level hierarchy, i.e., a plateau collection inside another one, the function is applied recursively in the lower levels until the quadruple is built. Hence, it simplifies the representation of complex plateau collection objects. The t-conorm considered in the aggregation is the `max` operator.

Value

A `pcollection` object consisting of a quadruple (`PLATEAUPOINT`, `PLATEAULINE`, `PLATEAUREGION`, `PLATEAUCOMPOSITION`).

References

Carniel, A. C.; Schneider, M. Spatial Data Types for Heterogeneously Structured Fuzzy Spatial Collections and Compositions. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2020), pp. 1-8, 2020.

Examples

```
# Point components
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((10 10), (9 8), (7 7))", 1)
pcp5 <- create_component("MULTIPOINT((0 0), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (2 3))", 0.4)
# Line components
lcp1 <- create_component("LINESTRING(0 0, 1 1.5)", 0.2)
lcp2 <- create_component("LINESTRING(1 3, 1 2, 2 0.5)", 0.5)
lcp3 <- create_component("LINESTRING(2 1.2, 3 1.6, 4 4)", 0.7)
lcp4 <- create_component("LINESTRING(1 1.5, 2 1.2)", 1.0)
lcp5 <- create_component("LINESTRING(-1 1, 2 2)", 0.9)
# Polygon components
rcp1 <- create_component("POLYGON((0 0, 1 4, 2 2, 0 0))", 0.4)
rcp2 <- create_component("POLYGON((2 0.5, 4 1, 4 0, 2 0.5))", 0.8)

# Creating plateau point objects
ppoint1 <- create_pgeometry(list(pcp1, pcp2, pcp3), "PLATEAUPOINT")
ppoint2 <- create_pgeometry(list(pcp4, pcp5), "PLATEAUPOINT")
ppoint3 <- create_pgeometry(list(pcp4, pcp5), "PLATEAUPOINT")
ppoint4 <- create_pgeometry(list(pcp6, pcp7), "PLATEAUPOINT")
# Creating plateau line objects
pline1 <- create_pgeometry(list(lcp1, lcp3), "PLATEAULINE")
pline2 <- create_pgeometry(list(lcp2, lcp4), "PLATEAULINE")
pline3 <- create_pgeometry(list(lcp5), "PLATEAULINE")
# Creating a plateau region objects
pregion <- create_pgeometry(list(rcp1, rcp2), "PLATEAUREGION")
# Creating a plateau composition object
pcomposition <- create_pgeometry(list(ppoint4, pline3), "PLATEAUCOMPOSITION")
# Creating plateau collection objects
pcol1 <- create_pgeometry(list(ppoint1, ppoint2, ppoint3, pline1), "PLATEAUCOLLECTION")
pcol2 <- create_pgeometry(list(pline2, pregion, pcomposition, pcol1), "PLATEAUCOLLECTION")
## Not run:
pcol2
plot(pcol2)

flatten_col <- spa_flatten(pcol2)
flatten_col
plot(flatten_col)

## End(Not run)
```

---

spa_get_type *Get the type of a spatial plateau object*

Description

spa_get_type() returns the type of a spatial plateau object. It can be either "PLATEAUPOINT", "PLATEAULINE", "PLATEAUREGION", "PLATEAUCOMPOSITION", or "PLATEAUCOLLECTION".

Usage

```
spa_get_type(pgo)
```

Arguments

pgo	A pgeometry object of any type.

Details

The spa_get_type() function yields the type of a spatial plateau object given as input. For instance, if the pgo is a object of the class ppoint (subclass of pgeometry), it returns "PLATEAUPOINT".

Value

The type of a spatial plateau object as a character object (i.e., a string).

Examples

```
pcomp1 <- create_component("MULTIPOINT(1 2, 3 2)", 0.4)
pcomp2 <- create_component("POINT(2 1)", 0.3)
ppoint <- create_pgeometry(list(pcomp1, pcomp2), "PLATEAUPOINT")

spa_get_type(ppoint)

lcomp1 <- create_component("LINESTRING(1 2, 3 3, 3 4)", 1)
lcomp2 <- create_component("LINESTRING(0 0, 5 5)", 0.5)
pline <- create_pgeometry(list(lcomp1, lcomp2), "PLATEAULINE")

spa_get_type(pline)

pcomposition <- create_pgeometry(list(ppoint, pline), "PLATEAUCOMPOSITION")

spa_get_type(pcomposition)
```

---

spa_is_empty *Check if a* pgeometry *object is empty*

Description

spa_is_empty() checks whether a given pgeometry object is empty (i.e., if it does not contain components).

Usage

```
spa_is_empty(pgo)
```

Arguments

pgo	A pgeometry object.

Details

The spa_is_empty() function checks if a pgeometry object has any component or not. If the number of components of a pgeometry object is equal to 0, then it returns TRUE. Otherwise, it returns FALSE.

Value

A Boolean value that indicates if a pgeometry is empty.

Examples

```
# Creating an empty plateau line object
pgo1 <- create_empty_pgeometry("PLATEAULINE")

# Checking if it is empty
spa_is_empty(pgo1)

# Adding a component to it and checking if it still empty
comp <- create_component("LINESTRING(1 1, 2 2, 2 3)", 0.5)
pgo1 <- spa_add_component(pgo1, comp)
spa_is_empty(pgo1)
```

---

spa_set_classification

Set a new classification for fuzzy topological relationships

Description

spa_set_classification() configures a new set of linguistic values and corresponding membership functions to be used by fuzzy topological relationships.

Usage

```
spa_set_classification(classes, mfs)
```

Arguments

classes	A character vector containing linguistic values that characterizes different situations of fuzzy topological relationships.
mfs	A vector of membership functions with domain in [0, 1].

Details

The `spa_set_classification()` function replaces the default linguistic values employed by fuzzy topological relationships. Each membership function *i* of the parameter `mfs` represents the class *i* of the parameter `classes`. The length of these parameters must to be equal.

Value

No return values, called for side effects.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of spatial plateau topological relationships and fuzzy topological relationships are respectively introduced in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.
- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.

Examples

```
## Not run:
library(tibble)
library(sf)

set.seed(456)

# Generating some random points to create pgeometry objects by using spa_creator()
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 30),
             z = runif(10, min = 0, max = 50))

# Getting the convex hull on the points to clip plateau region objects during their constructions
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, base_poly = ch, fuzz_policy = "fcp", k = 2)

plot(pregions$pgeometry[[1]])
plot(pregions$pgeometry[[2]])

# Showing results for spa_overlap() by considering default list of classes
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")

## End(Not run)
# Changing the default classification
```

```
trap_mf <- function(a, b, c, d) {
  function(x) {
    pmax(pmin((x - a)/(b - a), 1, (d - x)/(d - c), na.rm = TRUE), 0)
  }
}

classes <- c("superficially", "moderately", "completely")
superficially <- trap_mf(0, 0.2, 0.4, 0.6)
moderately <- trap_mf(0.4, 0.6, 0.8, 1)
completely <- trap_mf(0.6, 0.8, 1, 1)

spa_set_classification(classes, c(superficially, moderately, completely))
## Not run:
# Now the fuzzy topological relationships will use the new classification
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")

## End(Not run)
```

---

spa_support *Get the support of a* pgeometry *object*

Description

spa_support() yields a crisp spatial object (as an sfg object) that corresponds to the support of a pgeometry object given as input.

Usage

```
spa_support(pgo)
```

Arguments

pgo	A pgeometry object of any type.

Details

The spa_support() function employs the classical definition of *support* from the fuzzy set theory in the context of Spatial Plateau Algebra. The *support* only comprises the points with membership degree greater than or equal to 1. Hence, spa_support() returns the sfg object that represents the total extent of the pgeometry given as input. If the pgeometry is empty, then an empty sfg object is returned.

Value

An sfg object that represents the support of pgeometry. It can be an empty object, if pgeometry is empty.

References

Carniel, A. C.; Venâncio, P. V. A. B; Schneider, M. fsr: An R package for fuzzy spatial data handling. Transactions in GIS, vol. 27, no. 3, pp. 900-927, 2023.

Underlying concepts and formal definitions of Spatial Plateau Algebra are introduced in:

- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
pcp1 <- create_component("POINT(0 0)", 0.3)
pcp2 <- create_component("MULTIPOINT((2 2), (2 4), (2 0))", 0.5)
pcp3 <- create_component("MULTIPOINT((1 1), (3 1), (1 3), (3 3))", 0.9)
pcp4 <- create_component("MULTIPOINT((1 2), (2 1), (3 2))", 1)
pcp5 <- create_component("MULTIPOINT((0 0.5), (2 3))", 0.7)
pcp6 <- create_component("MULTIPOINT((0 1), (3 3.5))", 0.85)
pcp7 <- create_component("MULTIPOINT((1 0), (4 2))", 0.4)

# Creating a plateau point object
ppoint <- create_pgeometry(list(pcp1, pcp2, pcp3, pcp4, pcp5), "PLATEAUPOINT")
ppoint

# Getting its support
spa_support(ppoint)

# Getting the support of an empty pgeometry
spa_support(create_empty_pgeometry("PLATEAUREGION"))
```

---

visitation *Visitation: An example of FSI model*

Description

`visitation()` provides an example, without rules, of a fuzzy spatial inference (FSI) model.

Usage

```
visitation()
```

Details

The `visitation()` function provides a hypothetical FSI model that estimates the visiting experience based on prices and overall ratings of accommodations as well as sanitary conditions of restaurants. The output of such a model infers a value between 0 and 100 that indicates how attractive it is to visit a specific location. For this, the experience can be classified as *awful*, *average*, and *great*.

The linguistic variables and their linguistic values of this FSI model are listed below:

- *accommodation price* with *cut-rate*, *affordable*, and *expensive* as linguistic values.
- *accommodation review* with *bad*, *good*, and *excellent* as linguistic values.
- *food safety* with *low*, *medium*, and *high* as linguistic values, which represent levels of sanitary conditions.

Note that this is just a small running example, containing a small set of points to represent the locations of accommodations and restaurants.

The usage of FSI models is subdivided into a *preparation phase* and an *evaluation phase*. The preparation phase is responsible for instantiating a new FSI model with the elements of the data source component of FIFUS. For this, the `fsr` package provides the following functions: `fsi_create()`, `fsi_add_fsa()`, and `fsi_add_cs()`. These functions are employed by `visitation()` so that users can add their own fuzzy set rules (by using `fsi_add_rules()`) and perform the evaluation phase (by using the functions `fsi_eval()` and/or `fsi_qw_eval()`).

In this sense, `visitation()` performs the following internal actions to return an FSI model:

1. specify the linguistic variables and their corresponding linguistic values, which are in turn represented by membership functions. These items are specified according to the context of the running example.
2. define small point datasets that represent each linguistic variable. Such datasets are `tibble` objects.
3. build spatial plateau objects by using `spa_creator()` on the datasets. As a result, we get spatial plateau objects that represent each linguistic value.
4. create an FSI model with `fsi_create()` function.
5. add fuzzy spatial antecedents with `fsi_add_fsa()`. Recall that the antecedents are spatial plateau objects previously built.
6. define the linguistic variable and its linguistic values with membership functions for the consequent.
7. add the consequent to the FSI model by using `fsi_add_cs()`.

Value

An FSI model without fuzzy rules set.

References

This function is based on the running example introduced in:

- Carniel, A. C.; Galdino, F.; Philippsen, J. S.; Schneider, M. Handling Fuzzy Spatial Data in R Using the fsr Package. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems (AM SIGSPATIAL 2021), pp. 526-535, 2021.

Underlying concepts and formal definitions of FIFUS are discussed in:

- Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
fsi <- visitation()
```

Index