Package 'rSDI'

May 30, 2024

Type Package

Title Spatial Dispersion Index (SDI) Family of Metrics for Spatial/Geographic Networks

Version 0.2.1

URL https://github.com/ehengirmen/rSDI

BugReports https://github.com/ehengirmen/rSDI/issues

Maintainer Mehmet Gençer <mehmetgencer@yahoo.com>

Description Spatial Dispersion Index (SDI) is a generalized measurement index, or rather a family of indices to evaluate spatial dispersion of movements/flows in a network in a problem neutral way as described in: Gencer (2023) <doi:10.1007/s12061-023-09545-8>. This package computes and optionally visualizes this index with minimal hassle.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports igraph

RoxygenNote 7.3.1

Suggests maps, ggplot2, ggraph, ggimage, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Mehmet Gençer [aut, cre, cph] (<https://orcid.org/0000-0003-1717-8668>), Mücahit Zor [aut], Engin Hengirmen [aut]

Repository CRAN

Date/Publication 2024-05-30 07:40:02 UTC

R topics documented:

`dist_calc` *Distance Calculation for Graph Nodes*

Description

This function calculates distances between each pair of nodes in the graph. It supports both Haversine and Euclidean formula. The function automatically selects the formula based on the availabe vertex attributes: 'x' and 'y' for Euclidean distances, 'latitude' and 'longitude' for Haversine distances.

Usage

```
dist_calc(g, formula = NULL)
```

Arguments

`g`	An igraph object, with nodes that have a combination of 'latitude' and 'longitude', or 'x' and 'y' vertices attributes.
`formula`	Optional parameter to specify the distance calculation formula to use, either 'Haversine' or 'Euclidean'. By default 'formula = NULL' and if not specified the otherwise, the function automatically determines the formula based on the available vertex attributes. [x, y] => Euclidean, [latitude,longitude] => Haversine. Note that the 'g' must have one of this set of vertex attributes.

Value

An igraph object with an additional edge attribute 'distance' that contains the calculated distances between each pair of nodes.

Examples

```
# Assuming 'g' is a graph object with latitude and longitude or x and y attributes for each node.
# an overall example
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))

# user provides x and y vertices
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))

g<-igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)


dist_calc(g) # eucl. dist. calculated
dist_calc(g, formula = 'Euclidean') # calculates euc when asked
#dist_calc(g, formula = 'Haversine') # error

# user provides latitude and longitude vertices instead of x&y
nodes<-data.frame(id=c("A","B","C","D"),latitude=c(0,4,0,4),longitude=c(3,0,0,3))

g<-igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)

dist_calc(g) # haversine dist calculated
dist_calc(g, formula = 'Haversine') # calculated hav when asked specificallly
#dist_calc(g, formula = 'Euclidean') # error
```

---

euclidean *Calculate Euclidean Distance Between Two Points*

Description

This function calculates the Euclidean distance between two points. The Euclidean is the 'straight line' distance between two points in a two-dimensional space. This function takes the coordinates of two points (longitude and latitude) and calculates the straight distance between them, assuming flat Earth approximation.

Usage

```
euclidean(x1, y1, x2, y2)
```

Arguments

x1	X-coordinate of the first point.
y1	Y-coordinate of the first point.
x2	X-coordinate of the second point.
y2	Y-coordinate of the second point.

Value

The Euclidean distance between the two points.

Examples

```
euclidean(1, 2, 4, 6)
# Euclidean distance between points (1, 2) and (4, 6).
```

`haversine` *Calculate Haversine Distance Between Two Points on Earth*

Description

This function calculates the great-circle distance between two points on the Earth's surface, given their longitude and latitude in decimal degrees. It uses the Haversine formula, which accounts for the Earth's curvature.

Usage

```
haversine(lon1, lat1, lon2, lat2, R = 6371)
```

Arguments

`lon1`	Longitude of the first point in decimal degrees.
`lat1`	Latitude of the first point in decimal degrees.
`lon2`	Longitude of the second point in decimal degrees.
`lat2`	Latitude of the second point in decimal degrees.
`R`	The radius of the Earth in kilometers.

Value

The distance between the two points in kilometers.

Examples

```
haversine(-73.9851, 40.7580, -0.1278, 51.5074)
# Distance between NY City and London.
```

plotSDI *plotSDI generates a plot of the network and SDI metrics on a geographic map.*

Description

plotSDI generates a plot of the network and SDI metrics on a geographic map.

Usage

```
plotSDI(
  g,
  variant = "",
  circle.size.scale = 1,
  circle.color = "red",
  edges = FALSE,
  edge.width.range = c(0.01, 0.5)
)
```

Arguments

g	The igraph object to be plotted, whose vertices have attributes corresponding to SDI metrices.
variant	The SDI variant with a prefix, such as "SDI_vuw", etc.
circle.size.scale	Increase of decrease the size of circles drawn on nodes to represent SDI metric
circle.color	Change color of circles
edges	Whether to draw edges or not
edge.width.range	If edges are to be drawn give a custom range of edge widths

Value

returns a ggplot2 plot

Examples

```
TMSDI <- SDI(TurkiyeMigration.flows, TurkiyeMigration.nodes, variant="vuw")
plotSDI(TMSDI, variant="vuw", circle.size.scale=1)
```

SDI *Computes graph or vertex level Spatial Dispersion Index(ces).*

Description

If 'flows' is an igraph object then one can avoid supplying the nodes parameter. Alternatively one can supply flows as a data frame and nodes as another.

Usage

```
SDI(
  flows,
  nodes = NULL,
  distance.calculation = NULL,
  level = "vertex",
  weight.use = "weighted",
  directionality = "undirected",
  variant = NULL,
  alpha = NULL
)
```

Arguments

flows	A data frame or an igraph object
nodes	if flows are data frame, nodes must be supplied as a data frame. If flows are igraph object then not required
distance.calculation	optional method for distance calculation. 'Haversine' or 'Euclidean'. If not provided and edge distances are not available, distances are calculated by the SDI function.
level	The level to calculate the SDI. 'network' or 'vertex'
weight.use	'weighted', 'unweighted', or 'generalized'
directionality	'undirected', 'in', 'out', or 'all'
variant	Optional. Instead of specifying the level, directionality, and weight separately, the user can just supply a short-code of initial letters of each in that order to this argument, e.g. "vuw" for vertex level, undirected and weighted SDI. If it is a vector each value is treated separetely and multiple indices are computed.
alpha	Optional parameter used for generalized SDI calculations.

Details

To have an SDI computed you can must provide level, weight.use, and directionality parameters. Alternatively the 'variant' parameter can be specified which allows short-codes to indicate all of these three parameters. For example a value of "vui" for variant means a **v**ertex level,

undirected, and **i**nward directed SDI calculation. See the description of these three parameters to figure out possible short codes in a similar fashion.

The function returns an igraph object. If a network level calculation is requested the object will have an 'SDI_...' attribute whose name follows the short codes explained above. If a vertex level calculation is requested each vertex will have a similarly named attribute. For example the graph will have an 'SDI_nuw' attribute if variant is "nuw" (network level, undirected, and weighted). If variant is "vwu" each vertex will have an "SDI_vwu" attribute containing weighted undirected SDI for the vertex. If variant is a vector each value indicates a separate variant to be computed.

Value

An igraph object with SDI attributes added. The class of the object includes 'SDI'.

Examples

```
SDI(TurkiyeMigration.flows, TurkiyeMigration.nodes, variant="vuw")
```

`SDIcomputer` *SDIcomputer() is a helper function to compute given SDI variant for the given graph object. Not intended for explicit use. Called automatically by the 'SDI()' function.*

Description

SDIcomputer() is a helper function to compute given SDI variant for the given graph object. Not intended for explicit use. Called automatically by the 'SDI()' function.

Usage

```
SDIcomputer(g, level, weight.use, directionality, return.value = FALSE)
```

Arguments

`g`	An igraph object.
`level`	The level to calculate the SDI. 'network' or 'vertex'.
`weight.use`	'weighted' or 'unweighted'.
`directionality`	'undirected', 'in', 'out', or 'all'.
`return.value`	Logical. If TRUE, return the computed SDI value instead of modifying the graph object.

Value

If return.value is TRUE, returns the computed SDI value. Otherwise, returns the modified graph object.

Examples

```
TMgraph <- igraph::graph_from_data_frame(TurkiyeMigration.flows,
      directed=TRUE, TurkiyeMigration.nodes)
SDIcomputer(TMgraph,"vertex","weighted","in")
```

`TurkiyeMigration.flows`

Türkiye migration network flows data

Description

This data frame contains the data on migration of people between Türkiye's provinces in the period 2016-2017-2018. This is a consolidated version of raw data from Turkish Statistical Institute.

Usage

```
TurkiyeMigration.flows
```

Format

'TurkiyeMigration.flows' A data frame with 6480 rows and 3 columns:

from, to codes of origin and arrival province

weight number of people migrated

Details

Each row contains the number of people migrated in the "weight" column. 'from' and 'to' columns include the codes of provinces as used in the Türkiye statistical coding system.

The familiar names of provinces and their locations are to be found in a separate data frame named TurkiyeMigration.nodes

Source

<https://data.tuik.gov.tr/Kategori/GetKategori?p=Nufus-ve-Demografi-109>

TurkiyeMigration.nodes

Türkiye migration network provinces/nodes data

Description

This data frame contains the data on Türkiye's provinces as used in the migration flows data frame (TurkiyeMigration.flow)

Usage

```
TurkiyeMigration.nodes
```

Format

'TurkiyeMigration.nodes' A data frame with 6480 rows and 3 columns:

id codes of province as used in the TurkiyeMigration.flows data frame

label Name of the province capital city

longitude, latitude spatial coordinates of the province capital

Details

Each row contains the code of the province as used in TurkiyeMigration.flows data frame, it is known name/label, and latitude/longitude of the province central.

Source

The latitude/longitude of Turkish province capitals was shared as a courtesy of Başarsoft LLC

unweightedAllVerticesSDI

Not for explicit use.

Description

Not for explicit use.

Usage

```
unweightedAllVerticesSDI(g, mode = "all")
```

Arguments

g	the graph
mode	directionality 'undirected', 'in', 'out', or 'all'

Value

a vector of vertex SDI values

Examples

```
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))
toyGraph <- igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)
toyGraphWithSDI <- unweightedAllVerticesSDI(toyGraph)
```

---

unweightedNetworkSDI *Not for explicit use.*

Description

Not for explicit use.

Usage

```
unweightedNetworkSDI(g)
```

Arguments

g	the graph

Value

a numerical SDI value

Examples

```
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))
toyGraph <- igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)
toyGraphWithSDI <- unweightedNetworkSDI(toyGraph)
```

unweightedSingleVertexSDI *Not for explicit use.*

Description

Not for explicit use.

Usage

```
unweightedSingleVertexSDI(g, v, mode = "all")
```

Arguments

g	the graph
v	the vertex
mode	directionality 'undirected', 'in', 'out', or 'all'

Value

a numerical SDI value

Examples

```
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))
toyGraph <- igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)
toyGraph <- dist_calc(toyGraph)
toyGraphWithSDI <- unweightedSingleVertexSDI(toyGraph,igraph::V(toyGraph)[1])
```

variantParser *variantParser for SDI variant short-codes. This is a helper function and not intended for explicit use.*

Description

variantParser for SDI variant short-codes. This is a helper function and not intended for explicit use.

Usage

```
variantParser(variant)
```

Arguments

variant	a three letter short code for level, weight, and direction of SDI calculation.

Value

A list of explicit level, weight.use and directionality parameters

Examples

```
variantParser("vuw")
```

`weightedAllVerticesSDI` *Not for explicit use.*

Description

Not for explicit use.

Usage

```
weightedAllVerticesSDI(g, mode = "all")
```

Arguments

g	the graph
mode	directionality 'undirected', 'in', 'out', or 'all'

Value

a vector of vertex SDI values

Examples

```
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))
toyGraph <- igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)
toyGraphWithSDI <- weightedAllVerticesSDI(toyGraph)
```

weightedNetworkSDI *Not for explicit use.*

Description

Not for explicit use.

Usage

```
weightedNetworkSDI(g)
```

Arguments

g	the graph

Value

a numerical SDI value

Examples

```
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))
toyGraph <- igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)
toyGraphWithSDI <- weightedNetworkSDI(toyGraph)
```

weightedSingleVertexSDI *Not for explicit use.*

Description

Not for explicit use.

Usage

```
weightedSingleVertexSDI(g, v, mode = "all")
```

Arguments

g	the graph
v	the vertex
mode	directionality 'undirected', 'in', 'out', or 'all'

Value

a numerical SDI value

Examples

```
flows<-data.frame(from=c("A","B","A"), to=c("B","A","C"), weight=c(10,20,5))
nodes<-data.frame(id=c("A","B","C","D"),x=c(0,4,0,4),y=c(3,0,0,3))
toyGraph <- igraph::graph_from_data_frame(flows, directed=TRUE, vertices=nodes)
toyGraphWithSDI <- weightedSingleVertexSDI(toyGraph,igraph::V(toyGraph)[1])
```

Index