Package: rcoins (via r-universe)

May 19, 2026

Title Identify Naturally Continuous Lines in a Spatial Network

Version 0.4.0

Description Provides functionality to group lines that form naturally continuous lines in a spatial network. The algorithm implemented is based on the Continuity in Street Networks (COINS) method from Tripathy et al. (2021) <doi:10.1177/2399808320967680>, which identifies continuous ``strokes'' in the network as the line strings that maximize the angles between consecutive segments.

License Apache License (>= 2)

URL https://cityriverspaces.github.io/rcoins/, https://doi.org/10.5281/zenodo.14501804, https://github.com/CityRiverSpaces/rcoins

BugReports https://github.com/CityRiverSpaces/rcoins/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests ggplot2, knitr, rmarkdown, sfnetworks, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports dplyr, rlang, sf, sfheaders

VignetteBuilder knitr

Depends R (>= 4.1.0)

LazyData true

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

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

Date/Publication 2025-08-20 14:12:51 UTC

RemoteUrl https://github.com/cityriverspaces/rcoins

RemoteRef HEAD

RemoteSha 5554fddce5fe08f84d1ddce8f38d47c5de9e9dcf

Contents

get_example_data 2
stroke 2

Index **5**

`get_example_data` *Get example OSM data*

Description

This function retrieves example OpenStreetMap (OSM) data from the Zenodo data repository, and it can be used in examples and tests. The code used to generate the example dataset is available at https://github.com/CityRiverSpaces/CRiSpExampleData. Note that the example dataset is cached locally, so that subsequent calls to the function can load the example data from disk without having to re-download the data.

Usage

```
get_example_data()
```

Value

A list of sf objects containing the OSM data as `sf::sfc` objects.

Examples

```
get_example_data()
```

`stroke` *Identify naturally continuous lines in a spatial network*

Description

Provides functionality to group lines that form naturally continuous lines in a spatial network. The algorithm implemented is based on the Continuity in Street Networks (COINS) method doi:10.1177/2399808320967680, which identifies continuous "strokes" in the network as the line strings that maximize the angles between consecutive segments.

Usage

```
stroke(
  edges,
  angle_threshold = 0,
  attributes = FALSE,
  flow_mode = FALSE,
  from_edge = NULL
)
```

Arguments

edges	An object of class sfc (or compatible), including the network edge geometries (should be of type LINESTRING).
angle_threshold	Consecutive line segments can be considered part of the same stroke if the internal angle they form is larger than angle_threshold (in degrees). It should fall in the range 0 <= angle_threshold < 180.
attributes	If TRUE, return a label for each edge, representing the groups each edge belongs to. Only possible for flow_mode = TRUE.
flow_mode	If TRUE, line segments that belong to the same edge are not split across strokes (even if they form internal angles smaller than angle_threshold).
from_edge	Only look for the continuous strokes that include the provided edges or line segments.

Value

An object of class sfc (if attributes = FALSE), a vector with the same length as edges otherwise.

Examples

```
library(sf)

# Setup a simple network

p1 <- st_point(c(0, 3))
p2 <- st_point(c(2, 1))
p3 <- st_point(c(3, 0))
p4 <- st_point(c(1, 4))
p5 <- st_point(c(3, 2))
p6 <- st_point(c(4, 1))
p7 <- st_point(c(4, 3))
p8 <- st_point(c(5, 3))

l1 <- st_linestring(c(p1, p2, p5))
l2 <- st_linestring(c(p2, p3))
l3 <- st_linestring(c(p4, p5))
l4 <- st_linestring(c(p5, p6))
l5 <- st_linestring(c(p5, p7))
l6 <- st_linestring(c(p7, p8))
```

```
network_edges <- st_sfc(l1, l2, l3, l4, l5, l6)

# Identify strokes in the full network with default settings
stroke(network_edges)

# Set a threshold to the angle between consecutive segments
stroke(network_edges, angle_threshold = 150)

# Identify strokes in flow mode (do not break initial edges)
stroke(network_edges, flow_mode = TRUE)

# Instead of returning stroke geometries, return stroke labels
stroke(network_edges, flow_mode = TRUE, attributes = TRUE)

# Identify strokes that continue one (or a subset) of edges
stroke(network_edges, from_edge = 2)
stroke(network_edges, from_edge = c(2, 3))
```

Index

get_example_data, 2

sf::sfc, *2*
sfc, *3*
stroke, 2