Package: pathroutr (via r-universe)

May 11, 2026

Title An R Package for (Re-)Routing Paths Around Barriers

Version 0.2.2

Description The `pathroutr` package aims to provide a set of tools for routing marine animal tracks around land barriers based on the shortest path through a visibility graph network. The foundation of the package is a graph network created from a Delaunay Triangle mesh created from the vertices of land polygons within the study area. Any network edges that cross or fall completely within the land (barrier) polygons are removed.

License CC0

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports sf (>= 0.9), dplyr (>= 1.0), purrr, magrittr, units, sfnetworks, igraph, lwgeom, tibble, nabor

Suggests ggplot2, ggspatial, crawl, knitr, markdown, rmarkdown, remotes

Remotes NMML/crawl@devel

Depends R (>= 4.0)

VignetteBuilder knitr

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libglpk-dev libicu-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://ocean-tracking-network.r-universe.dev

Date/Publication 2026-03-11 19:00:52 UTC

RemoteUrl https://github.com/mhpob/pathroutr

RemoteRef HEAD

RemoteSha a0b615faea1b1f9119e55fd21e70637f62563a62

Contents

`akcoast` *Alaska coastline*

Description

Alaska 1:250000 coastal data polygon. This is provided by the Alaska Department of Natural Resources and was obtained from their open data portal (https://gis.data.alaska.gov/datasets/alaska-1250000). Note, only those polygons that intersect with the bounding box of our harbor seal movement are included.

Usage

```
akcoast
```

Format

Simple feature collection with 273 features and 5 fields:

geometry POLYGON

Source

https://gis.data.alaska.gov/datasets/alaska-1250000

get_barrier_segments *Identify track points that intersect with a barrier polygon*

Description

This function identifies the segments of consecutive points that intersect with the barrier polygon feature. The result is a tibble of segment records that identify portions of the track that will need to be re-routed.

Usage

```
get_barrier_segments(trkpts, barrier)
```

Arguments

trkpts	Simple Feature points ('sf', 'sfc_POINT'/'sfc_MULTIPOINT') that represent track points. Order is accepted as is and the bounding box of trkpts should be within the bounding box of the barrier polygon.
barrier	Simple Feature polygon ('sf', 'sfc_POLYGON'/'sfc_MULTIPOLYGON') representing the barrier feature. Should be the same barrier as supplied to the `prt_visgraph()` function.

Value

tibble representing segments of consecutive points that intersect with the barrier feature. the *start_pt* and *end_pt* geometry columns represent the bookend points for each segment that do not intersect with the barrier feature. The *n_pts* column is the number of points to be re-routed.

land_barrier *land barrier*

Description

A polygon dataset used to test and demonstrate package functions for routing paths around barriers

Usage

```
land_barrier
```

Format

Simple feature collection with 19 features and 0 fields:

geometry MULTIPOLYGON

Source

geopackage file available in extData

poi *points of interest*

Description

A point dataset used to test and demonstrate package functions for routing paths around barriers

Usage

```
poi
```

Format

Simple feature collection with 67 features and 0 fields:

geometry MULTIPOINT

Source

geopackage file available in extData

prt_extend_path *Extend a path to include given start and end points*

Description

This function extends the calculated shortest path to start and end with the provided *start_pt* and *end_pt* geometries. This ensures that the final sample of new points along the re-routed path include the portion of the line from the *start_pt*/*end_pt* to the nearest network node.

Usage

```
prt_extend_path(l_geom, start_pt, end_pt)
```

Arguments

l_geom	geometry passed from inside prt_shortpath()
start_pt	start point
end_pt	end point

Value

linestring

prt_nearestnode *Find the nearest node for start and end points in segs_tbl*

Description

Find the nearest node for start and end points in segs_tbl

Usage

```
prt_nearestnode(segs_tbl, vis_graph)
```

Arguments

segs_tbl	output from get_barrier_segments()
vis_graph	sfnetwork output from prt_visgraph()

Value

segs_tbl tibble with updated columns for nearest start and end nodes

prt_reroute *Re-route track points around barrier feature*

Description

This is a convenience wrapper, and the suggested function, for re-routing a *trkpts* series of ordered POINT features around a *barrier* polygon via *vis_graph* built with the prt_visgraph() function. The output can be used as a starting point for a custom process to replace the original geometry. Or, provide the output tibble directly to prt_update_points() along with *trkpts* for simply updating in place.

Usage

```
prt_reroute(trkpts, barrier, vis_graph, blend = TRUE)
```

Arguments

trkpts	Simple Feature points ('sf', 'sfc_POINT'/'sfc_MULTIPOINT') that represent track points. Order is accepted as is and the bounding box of trkpts should be within the bounding box of the barrier polygon.
barrier	Simple Feature polygon ('sf', 'sfc_POLYGON'/'sfc_MULTIPOLYGON') representing the barrier feature. Should be the same barrier as supplied to the prt_visgraph() function.
vis_graph	sfnetwork from prt_visgraph()
blend	boolean whether to blend start/end points into network

Details

The `blend = TRUE` argument will blend all of the *start_pt* / *end_pt* geometries in *segs_tbl* into the *vis_graph* network via the `sfnetworks::st_network_blend()` function. This process creates new nodes within *vis_graph* that are positioned at the perpendicular intersection between each point and the nearest edge. With `blend = FALSE` the nearest existing node is used. In highly complex coastlines, the use of `blend = FALSE` could result in re-routed paths that still intersect land or do not accurately represent the intended result. For less complex situations and when computational speed is important, `blend = FALSE` may be appropriate.

Value

a two-column tibble with column *fid* representing the row index in trkpts to be replaced by the new geometry in *geometry* column. If trkpts and barrier do not spatially intersect and empty tibble is returned.

---

prt_shortpath *Calculate the shortest path through a visibility network between two points*

Description

Given a *segs_tbl* tibble created by `get_barrier_segments()` and a *vis_graph* created by `prt_visgraph()`, this function adds a geometry column to *segs_tbl* that is the shortest path from the POINT geometries provided in the *start_pt* / *end_pt* columns of *segs_tbl*.

Usage

```
prt_shortpath(segs_tbl, vis_graph, blend = TRUE)
```

Arguments

`segs_tbl`	tibble from `get_barrier_segments()`
`vis_graph`	sfnetwork from prt_visgraph()
`blend`	boolean whether to blend start/end points into network

Details

The `blend = TRUE` argument will blend all of the *start_pt* / *end_pt* geometries in *segs_tbl* into the *vis_graph* network via the `sfnetworks::st_network_blend()` function. This process creates new nodes within *vis_graph* that are positioned at the perpendicular intersection between each point and the nearest edge. With `blend = FALSE` the nearest existing node is used. In highly complex coastlines, the use of `blend = FALSE` could result in re-routed paths that still intersect land or do not accurately represent the intended result. For less complex situations and when computational speed is important, `blend = FALSE` may be appropriate.

This function is typically called directly by `prt_reroute` and users are discouraged from using this function directly.

Value

segs_tbl data frame with added geometry column for shortest path LINESTRING that connects the *start_pt* and *end_pt* coordinates

---

prt_trim *Trim tracks to start and end outside barrier*

Description

Trim tracks to start and end outside barrier

Usage

```
prt_trim(trkpts, barrier)
```

Arguments

trkpts	Simple Feature points ('sf', 'sfc_POINT'/'sfc_MULTIPOINT') that represent track points. Order is accepted as is and the bounding box of trkpts should be within the bounding box of the barrier polygon.
barrier	Simple Feature polygon ('sf', 'sfc_POLYGON'/'sfc_MULTIPOLYGON') representing the barrier feature. Should be the same barrier as supplied to the `prt_visgraph()` function.

---

prt_update_points *Update track points with fixed geometry*

Description

Original geometry is updated in place and (currently) no record of those points that were updated is provided.

Usage

```
prt_update_points(rrt_pts, trkpts)
```

Arguments

rrt_pts	output from `prt_reroute()` or tibble with *rrt_idx* and *geometry* columns
trkpts	original trkpts Simple Features Collection

Value

trkpts with updated geometry

prt_visgraph *Create a visibility graph*

Description

Create a visibility graph

Usage

```
prt_visgraph(
  barrier,
  buffer = 0,
  centroids = FALSE,
  centroid_limit = 1e+07,
  aug_points = NULL
)
```

Arguments

barrier	simple feature 'POLYGON' or 'MULTIPOLYGON' that can be cast into 'POLYGON'
buffer	integer specifying buffer distance for barrier
centroids	logical whether to include centroids in the mesh
centroid_limit	integer minimum size (m^2) for adding centroid to triangles
aug_points	simple feature 'POINT' or 'MULTIPOINT' as additional nodes

Value

sfnetwork

spatial_predicates *Spatial predicates*

Description

These are custom spatial predicate functions that are negated versions of the spatial predicates `st_within()`, `st-crosses()`, and `st_intersects`

Usage

```
not_crosses(x, y)

not_within(x, y)

not_intersects(x, y)
```

Arguments

x, y	simple features.

Index