Package 'roads'

February 2, 2023

Title Road Network Projection

Version 1.1.0

Date 2023-03-11

URL https://github.com/LandSciTech/roads,
https://landscitech.github.io/roads/

Description Project road network development based on an existing road network, target locations to be connected by roads and a cost surface. Road projection methods include minimum spanning tree with least cost path (Kruskal's algorithm (1956) <doi:10.2307/2033241>), least cost path (Dijkstra's algorithm (1959) <doi:10.1007/BF01386390>) or snapping. These road network projection methods are ideal for use with land cover change projection models.

License Apache License (>= 2)

Encoding UTF-8

LazyData true

Imports raster, dplyr, igraph, sp, data.table, sf, units, rlang, methods, tidyselect, terra

RoxygenNote 7.2.1

Suggests testthat (>= 2.1.0), knitr, rmarkdown, viridis, tmap, bench

VignetteBuilder knitr

Depends R (>= 2.10)

Collate 'CLUSexample.R' 'buildSimList.R' 'buildSnapRoads.R' 'demoScen.R' 'getClosestRoad.R' 'getDistFromSource.R' 'getGraph.R' 'getLandingsFromTarget.R' 'lcpList.R' 'mstList.R' 'pathsToLines.R' 'projectRoads.R' 'rasterToLineSegments.R' 'shortestPaths.R' 'plotRoads.R' 'rasterizeLine.R'

BugReports https://github.com/LandSciTech/roads/issues

NeedsCompilation no

Author Sarah Endicott [aut, cre] (<https://orcid.org/0000-0001-9644-5343>),
Kyle Lochhead [aut],
Josie Hughes [aut],
Patrick Kirby [aut],
Her Majesty the Queen in Right of Canada as represented by the Minister of the Environment [cph] (Copyright holder for included functions buildSimList, getLandingsFromTarget, pathsToLines, plotRoads, projectRoads, rasterizeLine, rasterToLineSegments),
Province of British Columbia [cph] (Copyright holder for included functions getGraph, lcpList, mstList, shortestPaths, getClosestRoad, buildSnapRoads)

Maintainer Sarah Endicott <sarah.endicott@ec.gc.ca>

Repository CRAN

Date/Publication 2023-02-02 16:10:02 UTC

R topics documented:

`CLUSexample` *Data from the CLUS example*

Description

From Kyle Lochhead and Tyler Muhly's CLUS road simulation example

Usage

```
data(CLUSexample)
```

Format

A named list where: $cost is an object of class RasterLayer representing road cost $landings is an object of class SpatialPoints representing landing locations $roads is an object of class RasterLayer representing existing roads

demoScen *Demonstration set of 10 input scenarios*

Description

A demonstration set of scenarios that can be used as input to `projectRoads` method.

Usage

```
data(demoScen)
```

Format

A list of sub-lists, with each sub-list representing an input scenario. The scenarios (sub-lists) each contain the following components:

scen.number An integer value representing the scenario number (generated scenarios are numbered incrementally from 1).

road.rast A logical RasterLayer representing existing roads. TRUE is existing road. FALSE is not existing road.

road.line A SpatialLines object representing existing roads.

road.line.sf A sf object representing existing roads.

cost.rast A RasterLayer representing the cost of developing new roads on a given cell.

landings.points A SpatialPointsDataFrame representing landings sets and landing locations within each set. The data frame includes a field named 'set' which contains integer values representing the landings set that each point belongs to

landings.points.sf A sf object representing landings sets and landing locations within each set. The data frame includes a field named 'set' which contains integer values representing the landings set that each point belongs to

landings.stack A RasterStack representing the landings and landings sets. Each logical RasterLayer in the RasterStack represents one landings set. Values of TRUE are a landing in the given set. Values of FALSE are not.

landings.poly A SpatialPolygonsDataFrame representing a single set of polygonal landings.

landings.poly.sf A sf object representing a single set of polygonal landings.

See Also

`projectRoads`

`getDistFromSource` *Moving window approach to get distance from source*

Description

This function provides three different methods for calculating the distance of all points on a landscape from "source" locations. This is a computationally intensive process so the function arguments can be used to balance the tradeoffs between speed and accuracy. Note the pfocal versions are only available in the development version of the package.

Usage

```
getDistFromSource(src, maxDist, kwidth = 3)
```

Arguments

`src`	SpatRaster or RasterLayer, where all values > 0 are treated as source locations. NA values are treated as 0s.
`maxDist`	Numeric, maximum distance that should be calculated in units of the CRS.
`kwidth`	Integer, for the "pfocal" and "terra" methods the width of the moving window. For the "pfocal2" method the aggregation factor.

Details

The "terra" and "pfocal" methods use an iterative moving window approach and assign each cell a distance based on the number of times the moving window is repeated before it is included. This means that the moving window function is run many times but for a small window relative to the size of the raster. The ‘maxDist‘ argument determines the maximum distance calculated and affects the number of iterations of the moving window that are needed. ‘kwidth‘ is the radius of the moving window in number of cells, with larger values reducing the number of iterations needed but also reducing the granularity of the distances produced. The resulting distances will be in increments of ‘kwidth‘ * the resolution of the raster. The total number of iterations is ‘maxDist‘/ ‘kwidth‘ * resolution. The only difference in these methods is the underlying package used to do the moving window. The ‘terra‘ package has methods for handling large rasters by writing them to disk, while the ‘pfocal‘ package requires that the raster can be held in memory as a matrix.

The third method "pfocal2" uses a global moving window to calculate the distance to the source. This means that the moving window only needs to be applied once but the window size can be very large. In this case ‘maxDist‘ determines the total size of the window. ‘kwidth‘ can be used to reduce the number of cells included in the moving window by aggregating the source raster by a factor of ‘kwidth‘. This will increase the speed of computation but will produce results with artefacts of the larger grid and which may be less accurate since the output raster is disaggregated using bilinear interpolation.

Value

A SpatRaster

Examples

```
getDistFromSource(CLUSexample$roads, 5, 2)


 library(sf)
 library(terra)

#make example roads from scratch
rds <- data.frame(x = 1:1000/100, y = cos(1:1000/100)) %>%
  sf::st_as_sf(coords = c("x", "y")) %>%
  sf::st_union() %>%
  sf::st_cast("LINESTRING")

rds_rast <- terra::rasterize(terra::vect(rds),
                      terra::rast(nrows = 50, ncols = 50,
                           xmin = 0, xmax = 10,
                           ymin = -5, ymax = 5),
                      touches = TRUE)

getDistFromSource(rds_rast, 5, 2)
```

---

getLandingsFromTarget *Get landing points inside harvest blocks*

Description

Generate landing points inside polygons representing harvested area. There are three different sampling types available: "centroid" is the default and will return the centroid or a point that is inside the polygon if the centroid is not (see `st_point_on_surface`); "random" takes a random sample based on the given `landingDens` see (`st_sample`); "regular" intersects the polygons with a regular grid with cell size `sqrt(1/landingDens)`, if a polygon does not intersect with the grid its centroid is used.

Usage

```
getLandingsFromTarget(harvest, landingDens = NULL, sampleType = "centroid")
```

Arguments

`harvest`	sf, SpatialPolygons or RasterLayer object with harvested areas. If it is a RasterLayer with more than one unique value other than 0 each value will be run separately which will produce different results from a 0/1 raster but will be much slower.
`landingDens`	number of landings per unit area. This should be in the same units as the CRS of the harvest. Note that 0.001 points per m2 is > 1000 points per km2 so this number is usually very small for projected CRS.

`sampleType` character. "centroid" (default), "regular" or "random". Centroid returns one landing per harvest block, which is guaranteed to be in the harvest block for sf objects but not for rasters. Regular returns points from a grid with density `landingDens` that overlap the harvested areas. Random returns a random set of points from each polygon where the number is determined by the area of the polygons and `landingDens`. If `harvest` is a raster the centroid is always returned as one of the landings to ensure all harvest areas get at least one landing.

Details

Note that the `landingDens` is in points per unit area where the unit of area is determined by the CRS. For projected CRS this should likely be a very small number i.e. < 0.001.

Value

an sf simple feature collection with an ID column and POINT geometry

Examples

```
doPlots <- interactive()

polys <- demoScen[[1]]$landings.poly[1:2,]
# Get centroid
outCent <- getLandingsFromTarget(polys)

if(doPlots){
  raster::plot(polys)
  plot(outCent, col = "red", add = TRUE)
}

# Get random sample with density 0.1 points per unit area
outRand <- getLandingsFromTarget(polys, 0.1, sampleType = "random")

if(doPlots){
  raster::plot(polys)
  plot(outRand, col = "red", add = TRUE)
}

# Get regular sample with density 0.1 points per unit area
outReg <- getLandingsFromTarget(polys, 0.1, sampleType = "regular")

if(doPlots){
  raster::plot(polys)
  plot(outReg, col = "red", add = TRUE)
}
```

`plotRoads` *Plot projected roads*

Description

Plot the results of `projectRoads`

Usage

```
plotRoads(sim, mainTitle, subTitle = paste0("Method: ", sim$roadMethod), ...)
```

Arguments

`sim`	sim list result from `projectRoads`
`mainTitle`	A title for the plot
`subTitle`	A sub title for the plot, by default the `roadMethod` is used
`...`	Other arguments passed to raster plot call for the `costSurface`

Value

Creates a plot using base graphics

Examples

```
prRes <- projectRoads(CLUSexample$landings, CLUSexample$cost, CLUSexample$roads)
plotRoads(prRes, "Title")
```

`projectRoads` *Project road network*

Description

Project road locations based on existing roads, planned landings, and a cost surface that defines the cost of building roads.

Usage

```
projectRoads(
  landings = NULL,
  cost = NULL,
  roads = NULL,
  roadMethod = "mst",
  plotRoads = FALSE,
  mainTitle = "",
  neighbourhood = "octagon",
  sim = NULL,
  roadsOut = NULL,
  roadsInCost = TRUE,
  ordering = "closest"
)

## S4 method for signature 'ANY,ANY,ANY,ANY,ANY,ANY,ANY,missing'
projectRoads(
  landings = NULL,
  cost = NULL,
  roads = NULL,
  roadMethod = "mst",
  plotRoads = FALSE,
  mainTitle = "",
  neighbourhood = "octagon",
  sim = NULL,
  roadsOut = NULL,
  roadsInCost = TRUE,
  ordering = "closest"
)

## S4 method for signature 'ANY,ANY,ANY,ANY,ANY,ANY,ANY,list'
projectRoads(
  landings = NULL,
  cost = NULL,
  roads = NULL,
  roadMethod = "mst",
  plotRoads = FALSE,
  mainTitle = "",
  neighbourhood = "octagon",
  sim = NULL,
  roadsOut = NULL,
  roadsInCost = TRUE,
  ordering = "closest"
)
```

Arguments

landings	sf polygons or points, RasterLayer, SpatialPolygons*, SpatialPoints*, matrix, containing features to be connected to the road network. Matrix should contain columns x, y with coordinates, all other columns will be ignored.
cost	RasterLayer. Cost surface where existing roads must be the only cells with a cost of 0. If existing roads do not have 0 cost set `roadsInCost = FALSE` and they will be burned in.
roads	sf lines, SpatialLines*, RasterLayer. Existing road network.
roadMethod	Character. Options are "mst", "dlcp", "lcp", "snap".
plotRoads	Boolean. Should the resulting road network be plotted. Default FALSE.
mainTitle	Character. A title for the plot
neighbourhood	Character. 'rook','queen', or 'octagon'. The cells that should be considered adjacent. 'octagon' option is a modified version of the queen's 8 cell neighbourhood in which diagonals weights are 2^0.5x higher than horizontal/vertical weights.
sim	list. Returned from a previous iteration of `projectRoads`. cost, roads, and `roadMethod` are ignored if a `sim` list is provided.
roadsOut	Character. Either "raster", "sf" or NULL. If "raster" roads are returned as a raster in the `sim` list. If "sf" the roads are returned as an sf object which will contain lines if the roads input was sf lines but a geometry collection of lines and points if the roads input was a raster. The points in the geometry collection represent the existing roads while new roads are created as lines. If NULL (default) then the returned roads are sf if the input is sf or Spatial* and raster if the input was a raster.
roadsInCost	Logical. The default is TRUE which means the cost raster is assumed to include existing roads as 0 in its cost surface. If FALSE then the roads will be "burned in" to the cost raster with a cost of 0.
ordering	character. The order in which roads should be built to landings when 'roadMethod = "dlcp"'. Options are "closest" (default) where landings closest to existing roads are accessed first, or "none" where landings are accessed in the order they are provided in.

Details

Four different methods for projecting road networks have been implemented:

- "snap": Connects each landing directly to the closest road without reference to the cost or other landings
- "lcp": Least Cost Path connects each landing to the closest point on the road by determining the least cost path based on the cost surface provided, it does not consider other landings
- "dlcp": Dynamic Least Cost Path, same as "lcp" but it builds each path sequentially so that later roads will use earlier roads. The sequence of landings is determined by 'ordering' and is "closest" by default, the other option is "none" which will use the order that landings are supplied in.
- "mst": Minimum Spanning Tree connects all landings to the road by determining the least cost path to the road or other landings based on the cost surface

Value

a list with components:

- roads: the projected road network, including new and input roads.
- costSurface: the cost surface, updated to have 0 for new roads that were added.
- roadMethod: the road simulation method used.
- landings: the landings used in the simulation.
- g: the graph that describes the cost of paths between each cell in the cost raster. This is updated based on the new roads so that vertices were connected by new roads now have a cost of 0. This can be used to avoid recomputing the graph in a simulation with multiple time steps.

Examples

```
doPlots <- interactive()

projectRoads(CLUSexample$landings, CLUSexample$cost, CLUSexample$roads,
             "lcp", plotRoads = doPlots, mainTitle = "CLUSexample")


# More realistic examples that take longer to run

### using:  scenario 1 / sf landings / least-cost path ("lcp")
# demo scenario 1
scen <- demoScen[[1]]

# landing set 1 of scenario 1:
land.pnts <- scen$landings.points.sf[scen$landings.points.sf$set==1,]

prRes <- projectRoads(land.pnts, scen$cost.rast, scen$road.line.sf, "lcp",
                          plotRoads = doPlots, mainTitle = "Scen 1: SPDF-LCP")

### using: scenario 1 / RasterLayer landings / minimum spanning tree ("mst")
# demo scenario 1
scen <- demoScen[[1]]

# the RasterLayer version of landing set 1 of scenario 1:
land.rLyr <- scen$landings.stack[[1]]

prRes <- projectRoads(land.rLyr, scen$cost.rast, scen$road.line.sf, "mst",
                          plotRoads = doPlots, mainTitle = "Scen 1: Raster-MST")


### using: scenario 2 / matrix landings raster roads / snapping ("snap")
# demo scenario 2
scen <- demoScen[[2]]

# landing set 5 of scenario 2, as matrix:
land.mat  <- scen$landings.points[scen$landings.points$set==5,]@coords
```

```
prRes <- projectRoads(land.mat, scen$cost.rast, scen$road.rast, "snap",
                          plotRoads = doPlots, mainTitle = "Scen 2: Matrix-Snap")

## using scenario 7 / Polygon landings raster / minimum spanning tree
# demo scenario 7
scen <- demoScen[[7]]
# rasterize polygonal landings of demo scenario 7:
land.polyR <- raster::rasterize(scen$landings.poly, scen$cost.rast)

prRes <- projectRoads(land.polyR, scen$cost.rast, scen$road.rast, "mst",
                          plotRoads = doPlots, mainTitle = "Scen 7: PolyRast-MST")
```

`rasterizeLine` *Faster rasterize for lines*

Description

Rasterize a line using `stars` because `fasterize` doesn't work on lines and rasterize is slow. Deprecated use `terra::rasterize`

Usage

```
rasterizeLine(sfLine, rast, value)
```

Arguments

`sfLine`	an sf object to be rasterized
`rast`	a raster to use as template for the output raster
`value`	a number value to give the background ie 0 or NA

Value

a RasterLayer where the value of cells that touch the line will be the row index of the line in the sf

Examples

```
roadsLine <- sf::st_sf(geometry = sf::st_sfc(sf::st_linestring(
matrix(c(0.5, 4.5, 4.5, 4.51),
       ncol = 2, byrow = TRUE)
)))

# Deprecated rasterizeLine(roadsLine, CLUSexample$cost, 0)
# Use terra::rasterize
terra::rasterize(roadsLine, CLUSexample$cost, background = 0)
```

`rasterToLineSegments` *Convert raster to lines*

Description

Converts rasters that represent lines into an sf object.

Usage

```
rasterToLineSegments(rast, method = "mst")
```

Arguments

`rast`	raster representing lines all values > 0 are assumed to be lines
`method`	method of building lines. See Details

Details

For `method = "nearest"` raster is first converted to points and then lines are drawn between the nearest points. If there are two different ways to connect the points that have the same distance both are kept which can cause doubled lines. USE WITH CAUTION. `method = "mst"` converts the raster to points, reclassifies the raster so roads are 0 and other cells are 1 and then uses `projectRoads` to connect all the points with a minimum spanning tree. This will always connect all raster cells and is slower but will not double lines as often. Neither method is likely to work for very large rasters

Value

an sf simple feature collection

Examples

```
# works well for very simple roads
roadLine1 <- rasterToLineSegments(CLUSexample$roads)

# longer running more realistic examples

# mst method works well in this case
roadLine2 <- rasterToLineSegments(demoScen[[1]]$road.rast)

# nearest method has doubled line where the two roads meet
roadLine3 <- rasterToLineSegments(demoScen[[1]]$road.rast, method = "nearest")

# The mst method can also produce odd results in some cases
rasterToLineSegments(demoScen[[4]]$road.rast)
```

Index