

Package ‘roads’

January 16, 2026

Title Road Network Projection

Version 1.2.1

Date 2026-01-16

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

Description Iterative least cost path and minimum spanning tree methods for projecting forest road networks. The methods connect a set of target points to an existing road network using ‘igraph’ <<https://igraph.org>> to identify least cost routes. The cost of constructing a road segment between adjacent pixels is determined by a user supplied weight raster and a weight function; options include the average of adjacent weight raster values, and a function of the elevation differences between adjacent cells that penalizes steep grades. These road network projection methods are intended for integration into R workflows and modelling frameworks used for forecasting forest change, and can be applied over multiple time-steps without rebuilding a graph at each time-step.

License Apache License (>= 2)

Encoding UTF-8

LazyData true

Imports dplyr, igraph (>= 2.0.3), data.table, sf, units, rlang,
methods, tidyselect, terra

RoxygenNote 7.3.3

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

VignetteBuilder knitr

Depends R (>= 2.10)

Collate 'CLUSexample.R' 'buildSimList.R' 'buildSnapRoads.R'
'demoScen.R' 'getClosestRoad.R' 'getDistFromSource.R'
'getGraph.R' 'weightFunctions.R' 'getLandingsFromTarget.R'
'lcpList.R' 'mstList.R' 'pathsToLines.R' 'projectRoads.R'
'rasterToLineSegments.R' 'shortestPaths.R' 'plotRoads.R'
'rasterizeLine.R' 'prepExData.R' 'roads-package.R'
'dem_example.R'

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

NeedsCompilation no

Author Sarah Endicott [aut] (ORCID: <<https://orcid.org/0000-0001-9644-5343>>),

Kyle Lochhead [aut],

Josie Hughes [aut, cre],

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 Josie Hughes <josie.hughes@ec.gc.ca>

Repository CRAN

Date/Publication 2026-01-16 17:00:10 UTC

Contents

CLUSexample	2
demoScen	3
dem_example	4
getLandingsFromTarget	5
gradePenaltyFn	6
plotRoads	7
prepExData	8
projectRoads	9
rasterToLineSegments	13
simpleCostFn	14

Index

15

CLUSexample	<i>Data from the CLUS example</i>
-------------	-----------------------------------

Description

From Kyle Lochhead and Tyler Muhly's CLUS road simulation example. SpatRaster files created with the terra package must be saved with `terra::wrap()` and need to be unwrapped before they are used. `prepExData()` does this.

Usage

```
data(CLUSexample)
```

Format

A named list with components:

- cost: a PackedSpatRaster representing road building cost.
- landings: an sf dataframe of points representing landing locations.
- roads: a PackedSpatRaster representing existing roads.

Examples

```
CLUSexample  
prepExData(CLUSexample)
```

demoScen

Demonstration set of 10 input scenarios

Description

A demonstration set of scenarios that can be used as input to [projectRoads\(\)](#). The data contains SpatRaster objects that must be wrapped to be stored. To unwrap them use [prepExData\(\)](#)

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 PackedSpatRaster representing existing roads. TRUE is existing road. FALSE is not existing road.
- road.line: A sf object representing existing roads.
- cost.rast: A PackedSpatRaster representing the cost of developing new roads on a given cell.
- landings.points: 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 PackedSpatRaster with multiple layers representing the landings and landings sets. Each logical layer represents one landings set. Values of TRUE are a landing in the given set. Values of FALSE are not.
- landings.poly: A sf object representing a single set of polygonal landings.

See Also

[projectRoads](#)

Examples

```
demoScen[[1]]
demoScen <- prepExData(demoScen)
demoScen[[1]]
```

dem_example

Grade penalty example data

Description

A list containing two rasters covering an area near Revelstoke, British Columbia, Canada. ex_elev is elevation data and ex_wat is the proportion of the cell that contains water. Both are subsets of data downloaded with the geodata package at 30 arc seconds resolution. SpatRaster files created with the terra package must be saved with `terra::wrap()` and need to be unwrapped before they are used. `prepExData()` does this.

Usage

```
data(dem_example)
```

Format

A named list with components:

- ex_elev: a PackedSpatRaster of elevation.
- ex_wat: a PackedSpatRaster of proportion water.

Details

Elevation data are primarily from Shuttle Radar Topography Mission (SRTM), specifically the hole-filled CGIAR-SRTM (90 m resolution) from <https://srtm.csi.cgiar.org/>.

Water data are derived from the ESA WorldCover data set at 0.3-seconds resolution. (License CC BY 4.0). See <https://esa-worldcover.org/en> for more information.

References

Zanaga, D., Van De Kerchove, R., De Keersmaecker, W., Souverijns, N., Brockmann, C., Quast, R., Wevers, J., Grosu, A., Paccini, A., Vergnaud, S., Cartus, O., Santoro, M., Fritz, S., Georgieva, I., Lesiv, M., Carter, S., Herold, M., Li, Linlin, Tsednbazar, N.E., Ramoino, F., Arino, O., 2021. ESA WorldCover 10 m 2020 v100. doi:10.5281/zenodo.5571936.

Examples

```
dem_example
prepExData(dem_example)
```

```
getLandingsFromTarget  Get landing target points within harvest blocks
```

Description

Generate landing points inside polygons representing harvested area. There are three different sampling types available: "centroid" (default) returns the centroid or a point inside the polygon if the centroid is not (see `sf:::st_point_on_surface()`); "random" returns a random sample given `landingDens` see (`sf:::st_sample()`); "regular" returns points on a regular grid with cell size `sqrt(1/landingDens)` that intersect the polygon, or centroid if no grid points fall within the polygon.

Usage

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

Arguments

harvest	<code>sf</code> , <code>SpatialPolygons</code> , <code>SpatRaster</code> or <code>RasterLayer</code> object with harvested areas. If it is a raster with values outside 0,1, values are assumed to be harvest block IDs. If raster values are in 0,1 they are assumed to be a binary raster and <code>terra::patches</code> is used to identify harvest blocks.
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 m ² is > 1000 points per km ² 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 <code>sf</code> objects but not for rasters. "regular" returns points from a grid with density <code>landingDens</code> that overlap the harvested areas. "random" returns a random set of points from each polygon determined by the area of the polygon and <code>landingDens</code> . If <code>harvest</code> is a raster set of landings always includes the centroid to ensure at least one landing for each harvest block.

Details

Note that the `landingDens` is 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()
demoScen <- prepExData(demoScen)
```

```

polys <- demoScen[[1]]$landings.poly[1:2,]

# Get centroid
outCent <- getLandingsFromTarget(polys)

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

```

gradePenaltyFn

Grade penalty edge weight function

Description

Method for calculating the weight of an edge between two nodes from the value of the input raster at each of those nodes (x_1 and x_2), designed for a single DEM input. The method assumes an input `weightRaster` in which:

- NA indicates a road cannot be built
- Negative values are costs for crossing streams or other barriers that are crossable but expensive. Edges that link to barrier penalty (negative value) nodes are assigned the largest barrier penalty weight.
- Zero values are assumed to be existing roads.
- All other values are interpreted as elevation in the units of the raster map (so that a difference between two cells equal to the map resolution can be interpreted as 100% grade) This is a simplified version of the grade penalty approach taken by Anderson and Nelson (2004): The approach does not distinguish between adverse and favourable grades. Default construction cost values are from the BC interior appraisal manual. The approach ignores (unknown) grade penalties beside roads and barriers in order to avoid increased memory and computational burden associated with multiple input rasters.

Usage

```
gradePenaltyFn(
  x1,
  x2,
  hdistance,
  baseCost = 16178,
  limit = 20,
  penalty = 504,
  limitWeight = NA
)
```

Arguments

x1, x2	Number. Value of the input raster at two nodes.
hdistance	Number. Horizontal distance between nodes. hdistance, x1, and x2 should have the same units.
baseCost	Number. Construction cost of 0% grade road per km.
limit	Number. Maximum grade (%) on which roads can be built.
penalty	Number. Cost increase (per km) associated with each additional % increase in road grade.
limitWeight	Number. Value assigned to edges that exceed the grade limit. Try setting to a high (not NA) value if encountering problems with disconnected graphs.

References

Anderson AE, Nelson J (2004) Projecting vector-based road networks with a shortest path algorithm. Canadian Journal of Forest Research 34:1444–1457. <https://doi.org/10.1139/x04-030>

Examples

```
gradePenaltyFn(0.5,0.51,1)
gradePenaltyFn(0.5,0.65,1)
# grade > 20% so NA
gradePenaltyFn(0.5,0.75,1)
```

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	character. A title for the plot
subTitle	character. A sub title for the plot, by default the roadMethod is used
...	Other arguments passed to raster plot call for the weightRaster

Value

Creates a plot using base graphics

Examples

```
CLUSexample <- prepExData(CLUSexample)
prRes <- projectRoads(CLUexample$landings, CLUexample$cost, CLUexample$roads)
if(interactive()){
  plotRoads(prRes, "Title")
}
```

prepExData

Prepare example data

Description

Prepare example data included in the package that contain wrapped terra objects. This applies `terra::unwrap()` recursively to the list provided so that all PackedSpatRasters are converted to SpatRasters.

Usage

```
prepExData(x)
```

Arguments

x	list. Contains elements some of which are packed SpatRasters.
---	---

Value

The same list but with unwrapped SpatRasters

Examples

```
CLUSexample
prepExData(CLUexample)
```

projectRoads	<i>Project road network</i>
--------------	-----------------------------

Description

Project a road network that links target landings to existing roads. For all methods except "snap", a `weightRaster` and `weightFunction` together determine the cost to build a road between two adjacent raster cells.

Usage

```
projectRoads(  
  landings = NULL,  
  weightRaster = NULL,  
  roads = NULL,  
  roadMethod = "ilcp",  
  plotRoads = FALSE,  
  mainTitle = "",  
  neighbourhood = "octagon",  
  weightFunction = simpleCostFn,  
  sim = NULL,  
  roadsOut = NULL,  
  roadsInWeight = TRUE,  
  ordering = "closest",  
  roadsConnected = FALSE,  
  ...  
)  
  
## S4 method for signature 'ANY,ANY,ANY,ANY,ANY,ANY,ANY,ANY,missing'  
projectRoads(  
  landings = NULL,  
  weightRaster = NULL,  
  roads = NULL,  
  roadMethod = "ilcp",  
  plotRoads = FALSE,  
  mainTitle = "",  
  neighbourhood = "octagon",  
  weightFunction = simpleCostFn,  
  sim = NULL,  
  roadsOut = NULL,  
  roadsInWeight = TRUE,  
  ordering = "closest",  
  roadsConnected = FALSE,  
  ...  
)  
  
## S4 method for signature 'ANY,ANY,ANY,ANY,ANY,ANY,ANY,list'
```

```
projectRoads(
  landings = NULL,
  weightRaster = NULL,
  roads = NULL,
  roadMethod = "ilcp",
  plotRoads = FALSE,
  mainTitle = "",
  neighbourhood = "octagon",
  weightFunction = simpleCostFn,
  sim = NULL,
  roadsOut = NULL,
  roadsInWeight = TRUE,
  ordering = "closest",
  roadsConnected = FALSE,
  ...
)
```

Arguments

<code>landings</code>	sf polygons or points, RasterLayer, SpatialPolygons*, SpatialPoints*, or matrix. Contains features to be connected to the road network. Matrix should contain columns x, y with coordinates, all other columns will be ignored. Polygon and raster inputs will be processed by getLandingsFromTarget() to get the centroid of harvest blocks.
<code>weightRaster</code>	SpatRaster or RasterLayer. A weightRaster and weightFunction together determine the cost to build a road between two adjacent raster cells. For the default <code>weightFunction = simpleCostFn</code> , the weightRaster should specify the cost of construction across each raster cell. The value of cells that contain existing roads should be set to 0; if not set <code>roadsInWeight = FALSE</code> to adjust the cost of existing roads. To use the alternative grade penalty method, set <code>weightFunction = gradePenaltyFn</code> , and provide a weightRaster in which: <ul style="list-style-type: none"> • NA indicates a road cannot be built • Negative values are costs for crossing streams or other barriers that are crossable but expensive. • Zero values are existing roads. • All other values are interpreted as elevation in the units of the raster map (so that a difference between two cells equal to the map resolution can be interpreted as 100% grade).
<code>roads</code>	sf lines, SpatialLines*, RasterLayer, SpatRaster. The existing road network.
<code>roadMethod</code>	Character. Options are "ilcp", "mst", "lcp", and "snap". See Details below.
<code>plotRoads</code>	Boolean. Should the resulting road network be plotted. Default FALSE.
<code>mainTitle</code>	Character. A title for the plot.
<code>neighbourhood</code>	Character. "rook", "queen", or "octagon". Determines which cells are considered adjacent. The default "octagon" option is a modified version of the queen's 8 cell neighbourhood in which diagonal weights are multiplied by $2^{0.5}$.

weightFunction	function. Method for calculating the weight of an edge between two nodes from the value of the weightRaster at each of those nodes (x1 and x2). The default simpleCostFn is the mean. The alternative, gradePenaltyFn, sets edge weights as a function of the difference between adjacent weightRaster cells to penalize steep grades. Users supplying their own weightFunction should note that it must be symmetric, meaning that the value returned should not depend on the ordering of x1 and x2. The weightFunction must include arguments x1, x2 and
sim	list. Returned from a previous iteration of projectRoads. weightRaster, 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 SpatRaster 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 SpatRaster if the input was a raster.
roadsInWeight	Logical. If TRUE (default) the value of existing roads in the weightRaster is assumed to be 0. If FALSE cells in the weightRaster that contain existing roads will be set to 0.
ordering	character. The order in which landings are processed when roadMethod = "ilcp". 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.
roadsConnected	Logical. Are all roads fully connected? If TRUE and roadMethod = "mst" the MST graph can be simplified and the projection should be faster. Default is FALSE.
...	Optional additional arguments to weightFunction.

Details

Four road network projection methods are:

- "lcp": The Least Cost Path method connects each landing to the closest road with a least cost path, without reference to other landings.
- "ilcp": The Iterative Least Cost Path method iteratively connects each landing to the closest road with a least cost path, so that the path to each successive landing can include roads constructed to access previous landings. The sequence of landings is determined by ordering and is "closest" by default. The alternative "none" option processes landings in the order supplied by the user.
- "mst": The Minimum Spanning Tree method connects landings to the existing road with a minimum spanning tree that does not require users to specify the order in which landings are processed.
- "snap": Connects each landing to the closest (by Euclidean distance) road without, reference to the weights or other landings.

Value

a list with components:

- roads: the projected road network, including new and input roads.
- weightRaster: the updated weightRaster in which new and old roads have value 0.
- 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 updated weightRaster. Edges between vertices connected by new roads have weight 0. g can be used to avoid the cost of rebuilding the graph in a simulation with multiple time steps.

Examples

```
CLUSexample <- prepExData(CLUexample)
doPlots <- interactive()

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

# More realistic examples that take longer to run

demoScen <- prepExData(demoScen)

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

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

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

### using: scenario 1 / `SpatRaster` 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, "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 <- sf::st_coordinates(scen$landings.points[scen$landings.points$set==5,])

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 <- terra::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")

```

rasterToLineSegments *Convert raster to lines*

Description

Converts rasters that represent lines into an sf object.

Usage

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

Arguments

rast	SpatRaster. Raster representing lines all values > 0 are assumed to be lines
method	character. Method of building lines. Options are "mst" (default) or "nearest". See Details below.

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

```

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

# longer running more realistic examples

demoScen <- prepExData(demoScen)
# 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
roadLine4 <- rasterToLineSegments(demoScen[[4]]$road.rast)

```

simpleCostFn

Simple cost edge weight function

Description

Calculates the weight of an edge between two nodes as the mean value of an input cost raster at each of those nodes (x1 and x2).

Usage

```
simpleCostFn(x1, x2, hdistance)
```

Arguments

x1, x2	Number. Value of the input cost raster at two nodes.
hdistance	Number. Horizontal distance between the nodes - for penalizing longer diagonal edges.

Examples

```
simpleCostFn(0.5, 0.7, 1)
```

Index