Package 'leastcostpath'

May 15, 2020

Title Modelling Pathways and Movement Potential Within a Landscape

Version 1.3.6

Date 2020-05-13

Maintainer Joseph Lewis <josephlewis1992@gmail.com>

URL josephlewis.github.io

Description Provides functionality to calculate cost surfaces based on slope (e.g. Herzog, 2010; Llobera and Sluckin, 2007 <doi:10.1016/j.jtbi.2007.07.020>; París Roche, 2002; Tobler, 1993), traversing slope (Bell and Lock, 2000), and landscape features (Llobera, 2000) to be used when modelling pathways and movement potential within a landscape (e.g. Llobera, 2015; Verhagen, 2013; White and Barber, 2012 <doi:10.1016/j.jas.2012.04.017>).

Depends R (>= 3.4.0)

Imports gdistance (>= 1.2-2), raster (>= 2.6-7), rgdal (>= 1.3-3), rgeos (>= 0.3-28), sp (>= 1.3-1), parallel (>= 3.4-1), pbapply (>= 1.4-2), methods, stats

License GPL (>= 2)

Encoding UTF-8

LazyData true

Suggests knitr, rmarkdown, spdep (>= 1.1-3)

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation no

Author Joseph Lewis [aut, cre]

Repository CRAN

Date/Publication 2020-05-15 05:00:03 UTC

R topics documented:

`cost_matrix` *Create a cost based nearest neighbour matrix*

Description

Creates a cost based nearest neighbour matrix of k length for each provided location. This matrix can be used in the nb_matrix argument within the create_lcp_network function to calculate Least Cost Paths between origins and destinations.

Usage

```
cost_matrix(cost_surface, locations, k)
```

Arguments

`cost_surface`	`TransitionLayer` object (gdistance package). Cost surface to be used in calculating the k nearest neighbour
`locations`	`SpatialPoints`. Locations to calculate k nearest neighbours from
`k`	`numeric` number of nearest neighbours to be returned
	#' @return `matrix` cost-based k nearest neighbour for each location as specified in the locations argument. The resultant `matrix` can be used in the nb_matrix argument within the create_lcp_network function.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(nrow=50, ncol=50,  xmn=0, xmx=50, ymn=0, ymx=50,
crs='+proj=utm')

r[] <- stats::runif(1:length(r))

slope_cs <- create_slope_cs(r, cost_function = 'tobler')

locs <- sp::spsample(as(raster::extent(r), 'SpatialPolygons'),n=5,'regular')

matrix <- cost_matrix(slope_cs, locs, 2)

lcp_network <- create_lcp_network(slope_cs, locations = locs,
nb_matrix = matrix, cost_distance = FALSE, parallel = FALSE)
```

---

create_banded_lcps *Calculate Least Cost Paths from random locations within distances*

Description

Calculates Least Cost Paths from centre location to random locations within a specified distance band. This is based on the method proposed by Llobera (2015).

Usage

```
create_banded_lcps(
  cost_surface,
  location,
  min_distance,
  max_distance,
  radial_points,
  cost_distance = FALSE,
  parallel = FALSE
)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to be used in Least Cost Path calculation
location	SpatialPoints* (sp package). Location from which the Least Cost Paths are calculated. Only the first cell is taken into account
min_distance	numeric value. minimum distance from centre location
max_distance	numeric value. maximum distance from centre location
radial_points	numeric value. Number of random locations around centre location within distances

cost_distance	logical. if TRUE computes total accumulated cost for each Least Cost Path. Default is FALSE
parallel	logical. if TRUE, the Least Cost Paths will be calculated in parallel. Number of Parallel socket clusters is total number of cores available minus 1. Default is FALSE

Value

SpatialLinesDataFrame (sp package). The resultant object contains least cost paths (number of LCPs is dependent on radial_points argument) calculated from a centre location to random locations within a specified distance band.

Author(s)

Joseph Lewis

Examples

```
#r <- raster::raster(nrow=50, ncol=50,  xmn=0, xmx=50, ymn=0, ymx=50, crs='+proj=utm')

#r[] <- stats::runif(1:length(r))

#slope_cs <- create_slope_cs(r, cost_function = 'tobler')

#locs <- sp::spsample(as(raster::extent(r), 'SpatialPolygons'),n=1,'random')

#lcp_network <- create_banded_lcps(cost_surface = final_cost_cs, location = locs, min_distance = 20,
#max_distance = 50, radial_points = 10, cost_distance = FALSE, parallel = FALSE)
```

---

create_barrier_cs *Create Barrier Cost Surface*

Description

Creates a cost surface that incorporates barriers that inhibit movement in the landscape.

Usage

```
create_barrier_cs(raster, barrier, neighbours = 16)
```

Arguments

raster	RasterLayer (raster package). The Resolution, Extent, and Spatial Reference System of the provided RasterLayer is used when creating the resultant Barrier Cost Surface

barrier	Spatial* (sp package). Areas within the landscape that movement is inhibited. See details for more
neighbours	numeric value. Number of directions used in the Least Cost Path calculation. See Huber and Church (1985) for methodological considerations when choosing number of neighbours. Expected values are 4, 8, or 16. Default is 16

Details

The resultant Barrier Cost Surface is produced by assessing which areas of the raster coincide with the Spatial object as specified in the barrier argument. The areas of raster that coincide with the Spatial object are given a conductance value of 0, with all other areas given a Conductance value of 1. The conductance value of 0 ensures that movement is inhibited within these areas. Examples include rivers, lakes, and taboo areas.

Value

TransitionLayer (gdistance package) numerically expressing the barriers to movement in the landscape. The resultant TransitionLayer can be incorporated with other TransitionLayer through Raster calculations

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))
loc1 = cbind(2667670, 6479000)
loc1 = sp::SpatialPoints(loc1)

barrier <- create_barrier_cs(raster = r, barrier = loc1)
```

---

create_CCP_lcps *Calculate Cumulative Cost Paths from Radial Locations*

Description

Calculates Least Cost Paths from radial locations of a specified distance to the centre location. This is based on the method proposed by Verhagen (2013).

Usage

```
create_CCP_lcps(
  cost_surface,
  location,
  distance,
  radial_points,
  cost_distance = FALSE,
  parallel = FALSE
)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to be used in Least Cost Path calculation
location	SpatialPoints (sp package). Location to which the Least Cost Paths are calculated to. Only the first row is taken into account
distance	numeric value. Distance from centre location to the radial locations
radial_points	numeric value. Number of radial locations around centre location
cost_distance	logical. if TRUE computes total accumulated cost for each Least Cost Path. Default is FALSE
parallel	logical. if TRUE, the Least Cost Paths will be calculated in parallel. Number of Parallel socket clusters is total number of cores available minus 1. Default is FALSE

Value

SpatialLinesDataFrame (sp package). The resultant object contains least cost paths (number of LCPs is dependent on radial_points argument) calculated from radial locations to a centre location within a specified distance.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(nrow=50, ncol=50,  xmn=0, xmx=50, ymn=0, ymx=50,
crs='+proj=utm')

r[] <- stats::runif(1:length(r))

slope_cs <- create_slope_cs(r, cost_function = 'tobler')

locs <- sp::spsample(as(raster::extent(r), 'SpatialPolygons'),n=1,'regular')

lcp_network <- create_CCP_lcps(cost_surface = slope_cs, location = locs,
distance = 20, radial_points = 10, cost_distance = FALSE, parallel = FALSE)
```

---

create_cost_corridor *Create a Cost Corridor*

Description

Combines the accumulated cost surfaces from origin-to-destination and destination-to-origin to identify areas of preferential movement that takes into account both directions of movement.

Usage

```
create_cost_corridor(cost_surface, origin, destination, rescale = FALSE)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to be used in Cost Corridor calculation
origin	SpatialPoints* (sp package). orgin location from which the Accumulated Cost is calculated. Only the first cell is taken into account.
destination	SpatialPoints* (sp package). destination location from which the Accumulated Cost is calculated. Only the first cell is taken into account
rescale	logical. if TRUE raster values scaled to between 0 and 1. Default is FALSE

Value

RasterLayer (raster package). The resultant object is the accumulated cost surface from origin-to-destination and destination-to-origin and can be used to identify areas of preferential movement in the landscape.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))
slope_cs <- create_slope_cs(r, cost_function = 'tobler', neighbours = 16)

loc1 = cbind(2667670, 6479000)
loc1 = sp::SpatialPoints(loc1)

loc2 = cbind(2667800, 6479400)
loc2 = sp::SpatialPoints(loc2)

cost_corridor <- create_cost_corridor(slope_cs, loc1, loc2, rescale = FALSE)
```

---

create_feature_cs *Create a Landscape Feature cost surface*

Description

Creates a Landscape Feature Cost Surface representing the attraction/repulsion of a feature in the landscape. See Llobera (2000) for theoretical discussion in its application.

Usage

```
create_feature_cs(raster, locations, x, neighbours = 16)
```

Arguments

raster	RasterLayer (raster package). The Resolution, Extent, and Spatial Reference System of the provided RasterLayer is used when creating the resultant Barrier Cost Surface
locations	SpatialPoints* (sp package). Location of Features within the landscape
x	numeric vector. Values denoting the attraction/repulsion of the landscape features within the landscape
neighbours	numeric value. Number of directions used in the Least Cost Path calculation. See Huber and Church (1985) for methodological considerations when choosing number of neighbours. Expected values are 4, 8, or 16. Default is 16

Value

TransitionLayer (gdistance package) numerically expressing the attraction/repulsion of a feature in the landscape. The resultant TransitionLayer can be incorporated with other TransitionLayer through Raster calculations.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))
loc1 = cbind(2667670, 6479000)
loc1 = sp::SpatialPoints(loc1)

num <- seq(200, 1, length.out = 20)

feature <- create_feature_cs(raster = r, locations = loc1, x = num)
```

---

create_FETE_lcps *Calculate least cost paths from each location to all other locations.*

Description

Calculates least cost paths from each location to all other locations (i.e. From Everywhere To Everywhere (FETE)). This is based on the method proposed by White and Barber (2012).

Usage

```
create_FETE_lcps(
  cost_surface,
  locations,
  cost_distance = FALSE,
  parallel = FALSE
)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to be used in Least Cost Path calculation
locations	SpatialPoints* (sp package). Locations to calculate Least Cost Paths from and to
cost_distance	logical. if TRUE computes total accumulated cost for each Least Cost Path. Default is FALSE
parallel	logical. if TRUE the Least Cost Paths will be calculated in parallel. Number of Parallel socket clusters is total number of cores available minus 1. Default is FALSE

Value

SpatialLinesDataFrame (sp package). The resultant object contains least cost paths calculated from each location to all other locations

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(nrow=50, ncol=50,  xmn=0, xmx=50, ymn=0, ymx=50,
crs='+proj=utm')

r[] <- stats::runif(1:length(r))

slope_cs <- create_slope_cs(r, cost_function = 'tobler')

locs <- sp::spsample(as(raster::extent(r), 'SpatialPolygons'),n=5,'regular')

lcp_network <- create_FETE_lcps(cost_surface = slope_cs, locations = locs,
cost_distance = FALSE, parallel = FALSE)
```

---

create_lcp *Calculate Least Cost Path from Origin to Destination*

Description

Calculates a Least Cost Path from an origin location to a destination location. Applies Dijkstra's algorithm.

Usage

```
create_lcp(
  cost_surface,
  origin,
  destination,
  directional = FALSE,
  cost_distance = FALSE
)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to be used in Least Cost Path calculation
origin	SpatialPoints* (sp package) location from which the Least Cost Path is calculated. Only the first row is taken into account
destination	SpatialPoints* (sp package) location to which the Least Cost Path is calculated. Only the first row is taken into account
directional	logical. if TRUE Least Cost Path calculated from origin to destination only. If FALSE Least Cost Path calculated from origin to destination and destination to origin. Default is FALSE
cost_distance	logical. if TRUE computes total accumulated cost for each Least Cost Path. Default is FALSE

Value

SpatialLinesDataFrame (sp package) of length 1 if directional argument is TRUE or 2 if directional argument is FALSE. The resultant object is the shortest route (i.e. least cost) between origin and destination using the supplied TransitionLayer.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))

slope_cs <- create_slope_cs(r, cost_function = 'tobler')

traverse_cs <- create_traversal_cs(r, neighbours = 16)

final_cost_cs <- slope_cs * traverse_cs

loc1 = cbind(2667670, 6479000)
loc1 = sp::SpatialPoints(loc1)

loc2 = cbind(2667800, 6479400)
loc2 = sp::SpatialPoints(loc2)
```

```
lcps <- create_lcp(cost_surface = final_cost_cs, origin = loc1,
destination = loc2, directional = FALSE, cost_distance = FALSE)
```

`create_lcp_density` *Creates a cumulative Least Cost Path Raster*

Description

Cumulatively combines Least Cost Paths in order to identify routes of preferential movement within the landscape.

Usage

```
create_lcp_density(lcps, raster, rescale = FALSE)
```

Arguments

`lcps`	`SpatialLines*` (sp package). Least Cost Paths
`raster`	`RasterLayer` (raster package). This is used to derive the resolution, extent, and spatial reference system to be used when calculating the cumulative least cost path raster
`rescale`	`logical`. if TRUE raster values scaled to between 0 and 1. Default is FALSE

Value

`RasterLayer` (raster package). The resultant object is the cumulatively combined Least Cost Paths. This identifies routes of preferential movement within the landscape.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(nrow=50, ncol=50,  xmn=0, xmx=50, ymn=0, ymx=50, crs='+proj=utm')

r[] <- stats::runif(1:length(r))

slope_cs <- create_slope_cs(r, cost_function = 'tobler')

x1 <- c(seq(1,10), seq(11,25), seq(26,30))
y1 <- c(seq(1,10), seq(11,25), seq(26,30))
line1 <- sp::SpatialLines(list(sp::Lines(sp::Line(cbind(x1,y1)), ID='a')))

x2 <- c(seq(1,10), seq(11,25), seq(26, 30))
y2 <- c(seq(1,10), seq(11,25), rep(25, 5))
line2 <- sp::SpatialLines(list(sp::Lines(sp::Line(cbind(x2,y2)), ID='b')))
```

```
lcp_network <- rbind(line1, line2)

cumulative_lcps <- create_lcp_density(lcps = lcp_network, raster = r, rescale = FALSE)
```

`create_lcp_network` *Calculate least cost paths from specified origins and destinations*

Description

Calculates least cost paths from each origins and destinations as specified in the neighbour matrix.

Usage

```
create_lcp_network(
  cost_surface,
  locations,
  nb_matrix = NULL,
  cost_distance = FALSE,
  parallel = FALSE
)
```

Arguments

`cost_surface`	`TransitionLayer` (gdistance package). Cost surface to be used in Least Cost Path calculation.
`locations`	`SpatialPoints*` (sp package). Potential locations to calculate Least Cost Paths from and to.
`nb_matrix`	`matrix`. 2 column matrix representing the index of origins and destinations to calculate least cost paths between.
`cost_distance`	`logical`. if TRUE computes total accumulated cost for each Least Cost Path. Default is FALSE.
`parallel`	`logical`. if TRUE, the Least Cost Paths will be calculated in parallel. Number of Parallel socket clusters is total number of cores available minus 1. Default is FALSE.

Value

`SpatialLinesDataFrame` (sp package). The resultant object contains least cost paths calculated from each origins and destinations as specified in the neighbour matrix.

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(nrow=50, ncol=50,  xmn=0, xmx=50, ymn=0, ymx=50,
crs='+proj=utm')

r[] <- stats::runif(1:length(r))

slope_cs <- create_slope_cs(r, cost_function = 'tobler')

locs <- sp::spsample(as(raster::extent(r), 'SpatialPolygons'),n=5,'regular')

lcp_network <- create_lcp_network(slope_cs, locations = locs,
nb_matrix = cbind(c(1, 4, 2, 1), c(2, 2, 4, 3)), cost_distance = FALSE, parallel = FALSE)
```

`create_slope_cs` *Create a slope based cost surface*

Description

Creates a cost surface based on the difficulty of moving up/down slope. This function provides the choice of multiple isotropic and anisotropic cost functions that estimate human movement across a landscape. Maximum percentage slope possible for traversal can also be suppplied.

Usage

```
create_slope_cs(
  dem,
  cost_function = "tobler",
  neighbours = 16,
  crit_slope = 12,
  max_slope = NULL
)
```

Arguments

`dem`	`RasterLayer` (raster package). Digital Elevation Model
`cost_function`	`character`. Cost Function used in the Least Cost Path calculation. Implemented cost functions include 'tobler', 'tobler offpath', 'irmischer-clarke male', 'irmischer-clarke offpath male', 'irmischer-clarke female', 'irmischer-clarke offpath female', 'modified tobler', 'wheeled transport', 'herzog', 'llobera-sluckin', 'all'. Default is 'tobler'. See Details for more information
`neighbours`	`numeric` value. Number of directions used in the Least Cost Path calculation. See Huber and Church (1985) for methodological considerations when choosing number of neighbours. Expected values are 4, 8, or 16. Default is 16

`crit_slope` `numeric` value. Critical Slope (in percentage) is 'the transition where switch-backs become more effective than direct uphill or downhill paths'. Cost of climbing the critical slope is twice as high as those for moving on flat terrain and is used for estimating the cost of using wheeled vehicles. Default value is 12, which is the postulated maximum gradient traversable by ancient transport (Verhagen and Jeneson, 2012). Critical slope only used in 'wheeled transport' cost function

`max_slope` `numeric` value. Maximum percentage slope that is traversable. Slope values that are greater than the specified max_slope are given a conductivity value of 0. Default is NULL

Details

Tobler's 'Hiking Function' is the most widely used cost function when approximating the difficulty of moving across a landscape (Gorenflo and Gale, 1990; Wheatley and Gillings, 2001). The function assess the time necessary to traverse a surface and takes into account up-slope and down-slope (Kantner, 2004; Tobler, 1993).

Tobler's offpath Hiking Function reduces the speed of the Tobler's Hiking Function by 0.6 to take into account walking off-path (Tobler, 1993)

The Irmischer and Clark functions were modelled from speed estimates of United States Military Academy (USMA) cadets while they navigated on foot over hilly, wooded terrain as part of their summer training in map and compass navigation.

The Modified Hiking cost function combines MIDE (París Roche, 2002), a method to calculate walking hours for an average hiker with a light load (Márquez-Pérez et al. 2017), and Tobler's 'Hiking Function' (Tobler, 1993). The Modified Hiking Function benefits from the precision of the MIDE rule and the continuity of Tobler's Hiking Function (Márquez-Pérez et al. 2017).

Herzog (2013), based on the cost function provided by Llobera and Sluckin (2007), has provided a cost function to approximate the cost for wheeled transport. The cost function is symmetric and is most applicable for use when the same route was taken in both directions.

Herzog's (2010) Sixth-degree polynomial cost function approximates the energy expenditure values found in Minetti et al. (2002) but eliminates the problem of unrealistic negative energy expenditure values for steep downhill slopes.

Llobera and Sluckin (2007) cost function approximates the metabolic energy expenditure in KJ/(m*kg) when moving across a landscape.

Value

`TransitionLayer` (gdistance package) numerically expressing the difficulty of moving up/down slope based on the cost function provided in the cost_function argument. list of `TransitionLayer` if cost_function = 'all'

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))
slope_cs <- create_slope_cs(r, cost_function = 'tobler', neighbours = 16, max_slope = NULL)
```

`create_traversal_cs` *Create a Traversal across Slope Cost Surface*

Description

Creates a cost surface based on the difficulty of traversing across slope. Difficulty of traversal is based on the figure given in Bell and Lock (2000). Traversal across slope accounts for movement directly perpendicular across slope being easier than movement diagonally up/down slope.

Usage

```
create_traversal_cs(dem, neighbours = 16)
```

Arguments

`dem`	`RasterLayer` (raster package). Digital Elevation Model
`neighbours`	`numeric` value. Number of directions used in the Least Cost Path calculation. See Huber and Church (1985) for methodological considerations when choosing number of neighbours. Expected values are 4, 8, or 16. Default is 16

Value

`TransitionLayer` (gdistance package) numerically expressing the difficulty of moving across slope based on figure given in Bell and Lock (2000). The traversal_cs `TransitionLayer` should be multiplied by the create_slope_cs `TransitionLayer`, resulting in a `TransitionLayer` that takes into account movement across slope in all directions

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))
traversal_cs <- create_traversal_cs(r, neighbours = 16)
```

validate_lcp *Calculate accuracy of Least Cost Path*

Description

Calculates the accuracy of a Least Cost Path using the buffer method proposed by Goodchild and Hunter (1997).

Usage

```
validate_lcp(lcp, comparison, buffers = c(50, 100, 250, 500, 1000))
```

Arguments

lcp	SpatialLines* (sp package). Least Cost Path to assess the accuracy of. Expects object of class SpatialLines/SpatialLinesDataFrame
comparison	SpatialLines* to validate the Least Cost Path against.
buffers	numeric vector of buffer distances to assess. Default values are c(50, 100, 250, 500, 1000).

Value

data.frame (base package). The resultant object identifies the percentage of the lcp within x distance (as supplied in the buffers argument) from the provided comparison object.

Author(s)

Joseph Lewis

Examples

```
x1 <- c(1,5,4,8)
y1 <- c(1,3,4,7)
line1 <- sp::SpatialLines(list(sp::Lines(sp::Line(cbind(x1,y1)), ID='a')))
x2 <- c(1,5,5,8)
y2 <- c(1,4,6,7)
line2 <- sp::SpatialLines(list(sp::Lines(sp::Line(cbind(x2,y2)), ID='b')))

val_lcp <- validate_lcp(lcp = line1, comparison = line2, buffers = c(0.1, 0.2, 0.5, 1))
```

Index