Package 'leastcostpath'

October 13, 2022

Title Modelling Pathways and Movement Potential Within a Landscape

Version 1.8.7

Date 2022-05-22

Maintainer Joseph Lewis <josephlewis1992@gmail.com>

URL Available at: https://CRAN.R-project.org/package=leastcostpath

Description Calculates cost surfaces based on slope to be used when modelling pathways and movement potential within a landscape (Lewis, 2021) <doi:10.1007/s10816-021-09522-w>.

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, Matrix, gstat

License GPL-3

Encoding UTF-8

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

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Joseph Lewis [aut, cre]

Repository CRAN

Date/Publication 2022-06-03 07:50:06 UTC

R topics documented:

`add_dem_error` *Incorporate vertical error into Digital Elevation Model*

Description

Incorporates vertical error into the supplied Digital Elevation Model.

Usage

```
add_dem_error(dem, rmse, size = "auto", vgm_model = "Sph")
```

Arguments

`dem`	`RasterLayer` (raster package). Digital Elevation Model
`rmse`	`numeric`. Vertical Root Mean Square Error of the Digital Elevation Model
`size`	`character` or `numeric`. Size of window when applying mean filter to random error fields. Increasing the size of the window increases the spatial autocorrelation in the random error field. Size of window is automatically calculated via a variogram when argument is 'auto' (default). If size of window is user-supplied, then numeric value must be odd.
`vgm_model`	`character`. Variogram model type when determining window size. Accepted values are 'Sph' (default), 'Exp', 'Gau', 'Mat'. See details for more information

Details

Digital Elevation Models (DEMs) are representations of the earth's surface and are subject to error (Wechsler, 1999). However the impact of the error on the results of analyses is often not evaluated (Hunter and Goodchild, 1997; Wechsler, 1999).

The add_dem_error function incorporates vertical error into the supplied Digital Elevation Model by assuming that the error for each cell follows a gaussian (normal) distribution around the measured elevation value and the global Root Mean Square Error (RMSE) estimating the local error variance around this values (Fisher and Tate, 2006). Addition of spatial autocorrelation applied by using a mean-window filter based on a window size (Wechsler and Kroll, 2006). If size argument is 'auto' then window size calculated via a variogram (Wechsler and Kroll, 2006).

vgm_model is the model fitted to the observed DEM variogram. This is used to calculate the distance at which spatial autocorrelation is no longer present (i.e. the range). If the vgm model type is not able to converge, try another model type (e.g. 'Gau').

Examples of RMSE for various datasets:

Shuttle Radar Topography Mission (SRTM) has a RMSE of 9.73m

Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) Global Digital Elevation Model (GDEM) has a RMSE of 10.20m

Ordnance Survey OS Terrain 5 has a maximum RMSE of 2.5m

Ordnance Survey OS Terrain 50 has a maximum RMSE of 4m

Value

`raster` (raster package). Digital Elevation Model with a single realisation of vertical error incorporated

Author(s)

Joseph Lewis

References

Fisher, P. F., Tate, N. J. (2006). Causes and consequences of error in digital elevation models. Progress in Physical Geography, 30(4), 467-489.

Hunter, G. J., Goodchild, M. F. (1997). Modeling the uncertainty of slope and aspect estimates derived from spatial databases. Geographical Analysis, 29: 35-49.

Wechsler, S. P. (1999) Digital Elevation Model (DEM) uncertainty: evaluation and effect on topographic parameters In Proceedings of the 1999 ESRI User Conference (available at: https://proceedings.esri.com/library/userconf/proc99/proceed/papers/pap262/p262.htm)

Wechsler, S. P. (2003). Perceptions of Digital Elevation Model Uncertainty by DEM Users, URISA Journal, 15, 57-64.

Wechsler, S. P., Kroll, C. N. (2006). Quantifying DEM Uncertainty and its Effect on Topographic Parameters. Photogrammetric Engineering & Remote Sensing, 72(9), 1081-1090. doi: 10.14358/pers.72.9.1081

Wechsler, S. P. (2007). Uncertainties associated with digital elevation models for hydrologic applications: a review. Hydrology and Earth System Sciences, 11, 4, 1481-1500. doi: 10.5194/hess1114812007

Examples

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

r_error <- add_dem_error(r, rmse = 9.73, size = 'auto', vgm_model = 'Gau')
```

`apply_cost` *Apply Cost Function to Slope (rise over run) values*

Description

Creates a Conductivity surface based on the difficulty of moving up/down slope. This function applies the cost function to the slope (rise over run) values as calcualted using calculate_slope()

Usage

```
apply_cost(
  slope = slope,
  cost_function = "tobler",
  neighbours = 16,
  crit_slope = 12,
  max_slope = NULL,
  percentile = 0.5
)
```

Arguments

`slope`	`TransitionLayer` (gdistance package). Slope (rise over run) as calculated using calculate_slope()
`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' and 'campbell 2019'. 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 numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 16
`crit_slope`	`numeric` value. Critical Slope (in percentage) is 'the transition where switchbacks 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. If cost_function argument is 'campbell 2019' then max_slope is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. Default is NULL

`percentile` `numeric` value. Travel rate percentile only used in 'campbell 2019' cost_function. Expected numeric values are 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 0.99. Default is numeric value 0.50

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 assesses the time necessary to traverse a surface and takes into account up-slope and down-slope (Kantner, 2004; Tobler, 1993). Time unit measured in seconds.

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). Time unit measured in seconds.

The Irmischer and Clark cost functions (2018) 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. Time unit measured in seconds.

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). Time unit measured in seconds.

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 (J/(kg*m)) 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 (KJ/m) when moving across a landscape.

Campbell (2019) cost function (Lorentz distribution) approximates the time taken to traverse a surface based on crowdsourced GPS data (1.05 million travel rate records). Data divided into travel rate percentiles (1st, 5th to 95th, by 5, and 99th). max_slope argument is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. Time unit measured in seconds.

Value

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

Author(s)

Joseph Lewis

calculate_slope *calculate slope (rise over run) from supplied digital elevation model (DEM)*

Description

calculate slope (rise over run) from supplied digital elevation model (DEM)

Usage

```
calculate_slope(dem, neighbours, exaggeration)
```

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, 16, 32, or 48. Default is 16
exaggeration	logical. if TRUE, positive slope values (ie. up-hill movement) multiplied by 1.99 and negative slope values (ie. down-hill movement) multiplied by 2.31.

Value

TransitionMatrix (gdistance package). Anisotropic Slope (rise over run) Conductivity surface

Author(s)

Joseph Lewis

check_locations *Check locations*

Description

Checks that locations can be reached when calculating least cost paths

Usage

```
check_locations(cost_surface, locations)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to be used when checking whether supplied locations are traversable from at least one adjcacent cell
locations	SpatialPoints* (sp package) locations to check

Details

Using the supplied cost surface, the function checks whether the cells of supplied locations are traversable from at least one adjacent cell. If the cells of the supplied location are not traversable from at least one adjacent cell then a calculated least cost path cannot traverse to that location.

Value

`numeric vector` of location indexes that are not traversable from at least one adjacent cell

Author(s)

Joseph Lewis

---

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

Value

`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,
  ncores = 1
)
```

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. If there are multiple SpatialPoints in the supplied data, only the first SpatialPoint 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. Default is FALSE
ncores	numeric. Number of cores used if parallel is TRUE. Default value is 1.

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

References

Llobera, M. (2015). Working the digital: some thoughts from landscape archaeology. In Chapman R, Wylie A (eds), Material evidence: learning from archaeological practice (pp. 173-188). Abingdon: Routledge.

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 = slope_cs, location = locs, min_distance = 5,
max_distance = 25, 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, field = 0, background = 1)
```

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) or RasterLayer (raster package). Area 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 numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 16
field	numeric value or character 'mask'. Value assigned to cells that coincide with the barrier Spatial* or RasterLayer object. Default is numeric value 0. If RasterLayer object supplied in barrier and field is 'mask' then RasterLayer values are assigned to the barrier
background	numeric value. Value assigned to cells that do not coincide with the Spatial* or RasterLayer object. Default is numeric value 1

Details

The resultant Barrier Cost Surface is produced by assessing which areas of the raster coincide with the Spatial* or RasterLayer object as specified in the barrier argument. The areas of the raster that coincide with the barrier are given a conductance value of 0 (default value, with all other areas given a Conductance value of 1 (default value). The conductance value of 0 ensures that movement is inhibited within these areas. Examples of use include rivers, altitudes, and taboo areas. If a RasterLayer object is supplied in the barrier argument then all cells with a value NOT NA will be used as the barrier.

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'))

pt = cbind(2667670, 6479000)
pt = sp::SpatialPoints(pt)
polygon <- rgeos::gBuffer(spgeom = pt, width = 200)
raster::crs(pt) <- raster::crs(r)
raster::crs(polygon) <- raster::crs(r)

barrier_pt <- create_barrier_cs(raster = r, barrier = pt)
```

```
barrier_polygon <- create_barrier_cs(raster = r, barrier = polygon)

r2 <- r
ext <- raster::extent(2667500, 2667900, 6478800, 6479500)
cells <- unlist(raster::cellFromPolygon(object = r, p = as(ext, 'SpatialPolygons')))
r2[-cells] <- NA

barrier_ras <- create_barrier_cs(raster = r, barrier = r2)
```

`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,
  ncores = 1
)
```

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. If there are multiple SpatialPoints in the supplied data, only the first SpatialPoint 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. Default is FALSE
`ncores`	`numeric`. Number of cores used if parallel is TRUE. Default value is 1.

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

References

Verhagen, P. (2013). On the road to nowhere? Least cost paths, accessibility and the predictive modelling perspective. In Contreras F, Farjas M, Melero FJ (eds). Fusion of cultures. Proceedings of the 38th annual conference on computer applications and quantitative methods in archaeology, Granada, Spain, April 2010. (pp 383-389). Oxford: Archaeopress

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_distance_cs *Create a Distance based cost surface*

Description

Creates a cost surface based on the distance between neighbouring cells. Distance corrected for if neighbours value greater than 4 (diagonal distance greater than straight line distance). Distance units are derived from the maximum resolution of the supplied RasterLayer.

Usage

```
create_distance_cs(raster, neighbours = 16)
```

Arguments

raster	RasterLayer (raster package).
neighbours	numeric value. Number of neighbouring cells. See Huber and Church (1985) for methodological considerations when choosing number of neighbours. Expected numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 16

Value

TransitionLayer (gdistance package) numerically expressing the distance between neighbouring cells

Author(s)

Joseph Lewis

Examples

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

---

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. Each value in the vector is assigned to each ring of cells moving outwards from supplied locations
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 numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 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

References

Llobera, M. (2000). Understanding movement: a pilot model towards the sociology of movement. In: Lock G (ed) Beyond the map. Archaeology and spatial technologies. (pp 66-84). Amsterdam: IOS Press/Ohmsha.

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,
  ncores = 1
)
```

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. Default is FALSE
ncores	numeric. Number of cores used if parallel is TRUE. Default value is 1.

Value

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

Author(s)

Joseph Lewis

References

White, DA. Barber, SB. (2012). Geospatial modeling of pedestrian transportation networks: a case study from precolumbian Oaxaca, Mexico. J Archaeol Sci 39:2684-2696. doi: 10.1016/j.jas.2012.04.017

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

References

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik. 1: 269-271.

Examples

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

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

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

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

lcps <- create_lcp(cost_surface = slope_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, rasterize_as_points = TRUE)
```

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
rasterize_as_points	logical. if TRUE (default) then the coordinates of the Least Cost Path vertices are rasterised. If FALSE Least Cost Paths are represented as lines and rasterised. As the Least Cost Path SpatialLines are converted from vector to raster, the Least Cost Paths represented as lines may result in the width of the rasterized line being greater than one cell, particularly at places of diagonal movement. Conversely, the Least Cost Paths represented as points (default) will result in some raster cells not being counted in the resultant RasterLayer. A greater number of cells not counted is expected when the number of neighbours used when creating the cost surface increases. NOTE: rasterisation of Lines takes much longer than rasterizing points.

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,
  ncores = 1
)
```

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. Default is FALSE
ncores	numeric. Number of cores used if parallel is TRUE. Default value is 1.

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 supplied. Lastly, geographical slant exaggeration can be accounted for.

Usage

```
create_slope_cs(
  dem,
  cost_function = "tobler",
  neighbours = 16,
  crit_slope = 12,
  max_slope = NULL,
  percentile = 0.5,
  exaggeration = FALSE
)
```

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' and 'campbell 2019'. 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 numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 16
crit_slope	numeric value. Critical Slope (in percentage) is 'the transition where switchbacks 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. If cost_function argument is 'campbell 2019' then max_slope is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. Default is NULL
percentile	numeric value. Travel rate percentile only used in 'campbell 2019' cost_function. Expected numeric values are 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 0.99. Default is numeric value 0.50
exaggeration	logical. if TRUE, positive slope values (ie. up-hill movement) multiplied by 1.99 and negative slope values (ie. down-hill movement) multiplied by 2.31. Based on how observers overestimate the slant of a hill. See Details for more information

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 assesses the time necessary to traverse a surface and takes into account up-slope and down-slope (Kantner, 2004; Tobler, 1993). Time unit measured in seconds.

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). Time unit measured in seconds.

The Irmischer and Clark cost functions (2018) 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. Time unit measured in seconds.

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). Time unit measured in seconds.

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 (J/(kg*m)) 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 (KJ/m) when moving across a landscape.

Campbell (2019) cost function (Lorentz distribution) approximates the time taken to traverse a surface based on crowdsourced GPS data (1.05 million travel rate records). Data divided into travel rate percentiles (1st, 5th to 95th, by 5, and 99th). max_slope argument is fixed at 30 degrees slope to reflect the maximum slope that the cost function is parametised to. Time unit measured in seconds.

Exaggeration

When observers face directly toward a hill, their awareness of the slant of the hill is greatly overestimated (Pingel, 2009; Proffitt, 1995; Proffitt, 2001). Pingel (2009) identified that downhill slopes are overestimated at approximately 2.3 times, whilst uphill slopes are overestimated at 2 times.

Value

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

Author(s)

Joseph Lewis

Examples

```
r <- raster::raster(system.file('external/maungawhau.grd', package = 'gdistance'))
slope_cs_16 <- create_slope_cs(r, cost_function = 'tobler', neighbours = 16, max_slope = NULL)
slope_cs_48 <- create_slope_cs(r, cost_function = 'tobler', neighbours = 48, max_slope = NULL)
```

`create_stochastic_lcp` *Calculate Stochastic Least Cost Path from Origin to Destination*

Description

Calculates a Stochastic Least Cost Path from an origin location to a destination location by randomly determining the neighbourhood adjacency. Method based on Pinto and Keitt (2009). Applies Dijkstra's algorithm. See details for more information.

Usage

```
create_stochastic_lcp(
  cost_surface,
  origin,
  destination,
  directional = FALSE,
  percent_quantile
)
```

Arguments

`cost_surface`	`TransitionLayer` (gdistance package). Cost surface to be used in Least Cost Path calculation. Threshold value applied to cost surface before calculating least cost path
`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
`percent_quantile`	`numeric`. Optional numeric value between 0 and 1. If argument is supplied then threshold is a random value between the minimum value in the supplied cost surface and the corresponding percent quantile value in the supplied cost surface. If no argument is supplied, then the threshold is a random value between the minimum value and maximum valie in the supplied cost surface. See details for more information

Details

The calculation of a stochastic least cost path is based on the method proposed by Pinto and Keitt (2009). Instead of using a static neighbourhood (for example as supplied in the neighbours function in the create_slope_cs), the neighbourhood is redefined such that the adjacency is non-deterministic and is instead determined randomly based on the threshold value.

The algorithm proceeds as follows:

1. If threshold_quantile is not supplied, draw a random value from a uniform distribution between the minimum value and maximum value in the supplied cost surface. If threshold_quantile is supplied, draw a random value between the minimum value in the supplied cost surface and the percent quantile as calculated using the supplied percent_quantile

2. Replace values in cost surface below the random value with 0. This ensures that the conductance between the neighbours are 0, and thus deemed non-adjacent.

Supplying a percent_quantile of 0 is equivalent to calculating the non-stochastic least cost path. That is, if the supplied percent_quantile is 0, then no values are below this value and thus no values will be replaced with 0 (see step 2). This therefore does not change the neigbourhood adjacency.

Supplying a percent_quantile of 1 is equivalent to not supplying a percent_quantile value at all. That is, if the supplied percent_quantile is 1, then the possible random threshold value is between the minimum and maximum values in the cost surface.

The closer the percent_quantile is to 0, the less the stochastic least cost paths are expected to deviate from the least cost path. For example, a percent_quantile value of 0.2 will result in the threshold being a random value between the minimum value in the cost surface and the 0.2 percent quantile of the values in the cost surface. All values in the cost surface below the threshold will be replaced with 0 (i.e. the neighbours are no longer adjacent). In contrast, a percent_quantile value of 0.8 will result in the threshold being a random value between the minimum value in the cost surface and the 0.8 percent quantile of the values in the cost surface. In this case, there is greater probability that the random value will result in an increased number of values in the cost surface being replaced with 0.

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 after a random threshold has been applied to the supplied `TransitionLayer`.

Author(s)

Joseph Lewis

References

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik. 1: 269-271.

Pinto, N., Keitt, T.H. (2009) Beyond the least-cost path: evaluating corridor redundancy using a graph-theoretic approach. Landscape Ecol 24, 253-266 doi: 10.1007/s109800089303y

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=2,'random')

stochastic_lcp <- create_stochastic_lcp(cost_surface = slope_cs,
origin = locs[1,], destination = locs[2,], directional = FALSE)
```

`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 numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 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)
```

`create_wide_lcp` *Calculate wide least cost path*

Description

Calculates a wide least cost path from an origin location to a destination location. Applies Dijkstra's algorithm. See details for more information

Usage

```
create_wide_lcp(
  cost_surface,
  origin,
  destination,
  neighbours = 16,
  path_ncells
)
```

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
`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 numeric values are 4, 8, 16, 32, 48 or a matrix object. Default is numeric value 16
`path_ncells`	`numeric` value. Dimension of wide path matrix. Note that the value refers to the number of cells and not distance. See `wide_path_matrix` for example

Details

The calculation of a wide least cost path is inspired by Shirabe (2015).Instead of calculating a least cost path where the path width is assumed to be zero or negligible compared to the cell size, create_wide_lcp creates a wide least cost path where the path is calculated based on a cost surface that incorporates the total permeability of passage from adjacent cells

The algorithm proceeds as follows:

Each column of the supplied cost surface is summed, resulting in a raster with each cell representing the total permeability of passage from each adjacent neighbour (adjacent cells specificed when creating cost surface through the use of wide_path_matirx(). A transitionMatrix is created from this total permeability of passage raster, with the permeability of movement between cells based on the

total permeability raster. That is, moving into each cell regardless of direction will incur the same cost.

Using this total permeability of passage cost surface, the least cost path can be calculated. This represents the least cost path between two locations based on the total permeability of passage cost surface that incorporates the summed permeability of passage. To visualise the wide least cost path, the least cost path is represented as a polygon with the width as supplied in the path_ncells argument.

Value

`SpatialPolygons` (sp package). The resultant object is the shortest wide path route (i.e. least cost) between origin and destination

Author(s)

Joseph Lewis

References

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik. 1: 269-271.

Shirabe, T. (2015). A method for finding a least-cost wide path in raster space. International Journal of Geographical Information Science 30, 1469-1485. doi: 10.1080/13658816.2015.1124435

Examples

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

n <- 3

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

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

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

lcps <- create_wide_lcp(cost_surface = slope_cs, origin = loc1,
destination = loc2, path_ncells = n)
```

---

crop_cs *Crop Cost Surface*

Description

Crops Cost Surfaces to the supplied SpatialPolygon* boundary

Usage

```
crop_cs(cost_surface, boundary)
```

Arguments

cost_surface	TransitionLayer (gdistance package). Cost surface to crop
boundary	Spatial* (sp package) or RasterLayer (raster package). Boundary used when cropping Cost Surface. See details for more

Details

The resultant Cost Surface is cropped to the Spatial* or RasterLayer object. All areas of the Cost Surface that are outside the supplied boundary are given a conductance value of 0. The conductance value of 0 ensures that movement is inhibited within these areas. If a RasterLayer object is supplied in the boundary argument then all cells with a value of NA will be given a Conductance value of 0.

Value

TransitionLayer (gdistance package). Cropped Cost Surface

Author(s)

Joseph Lewis

Examples

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

pt = cbind(2667670, 6479000)
pt = sp::SpatialPoints(pt)
polygon <- rgeos::gBuffer(spgeom = pt, width = 200)
raster::crs(pt) <- raster::crs(r)
raster::crs(polygon) <- raster::crs(r)

slope_cs <- create_slope_cs(r, cost_function = 'tobler', neighbours = 16, max_slope = NULL)

slope_cs_pt <- crop_cs(cost_surface = slope_cs, boundary = pt)
slope_cs_polygon <- crop_cs(cost_surface = slope_cs, boundary = polygon)

r2 <- r
ext <- raster::extent(2667500, 2667900, 6478800, 6479500)
cells <- unlist(raster::cellFromPolygon(object = r, p = as(ext, 'SpatialPolygons')))
r2[-cells] <- NA

slope_cs_raster <- crop_cs(cost_surface = slope_cs, boundary = r2)
```

`force_isotropy` *Convert anisotropic cost surfaces to isotropic*

Description

Averages transition values from-to adjacent cells

Usage

```
force_isotropy(cost_surface)
```

Arguments

`cost_surface`	`TransitionLayer` (gdistance package). Conductance surface

Details

force_isotropy averages (mean) the transition values from-to adjacent cells in a cost_surface. Through this, anisotropic cost functions (i.e. where movement down-slope is easier than movement up-slope) are converted to an isotropic cost function. When calculating an least cost path using the resultant surface, the least cost path from A-B and B-A will be the same. This is in contrast to anisotropic cost surfaces where the least cost path from A-B and B-A can differ.

Value

`TransitionLayer` (gdistance package) Conductance surface where transition values from-to adjacent cells have been averaged

Author(s)

Joseph Lewis

References

Herzog, I (2020). Spatial Analysis Based On Cost Functions, in Gillings, M., Hacıgüzeller, P., Lock, G. Archaeological Spatial Analysis. Routledge. pp. 333-358. doi: 10.4324/978135124385818

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)

slope_cs_iso <- force_isotropy(slope_cs)
```

neighbours_32 *32 Neighbourhood matrices based on Kovanen and Sarjakoski (2015)*

Description

see leastcostpath::neighbours_32 for layout

Usage

```
neighbours_32
```

Format

An object of class `matrix` (inherits from `array`) with 7 rows and 7 columns.

Author(s)

Joseph Lewis

References

Kovanen, J., Sarjakoski, T. (2015). Tilewise Accumulated Cost Surface Computation with Graphics Processing Units. ACM Transactions on Spatial Algorithms and Systems 1, 1-27. doi: 10.1145/2803172

neighbours_48 *48 Neighbourhood matrices based on Kovanen and Sarjakoski (2015)*

Description

see leastcostpath::neighbours_48 for layout

Usage

```
neighbours_48
```

Format

An object of class `matrix` (inherits from `array`) with 9 rows and 9 columns.

Author(s)

Joseph Lewis

References

Kovanen, J., Sarjakoski, T. (2015). Tilewise Accumulated Cost Surface Computation with Graphics Processing Units. ACM Transactions on Spatial Algorithms and Systems 1, 1-27. doi: 10.1145/2803172

`PDI_validation` *Calculate Path Deviation Index*

Description

Calculates the Path Deviation Index of a Least Cost Path and a comparison SpatialLines using the method proposed by Jan et al. (1999).

Usage

```
PDI_validation(lcp, comparison)
```

Arguments

`lcp`	`SpatialLines*` (sp package). Least Cost Path to assess the accuracy of. Expects object of class SpatialLines. Only first feature used.
`comparison`	`SpatialLines*` to validate the Least Cost Path against. Expects object of class SpatialLines. Only first feature used.

Details

The Path Deviation Index measures the deviation (i.e. the spatial separation) between a pair of paths and aims to overcome the shortcomings of measuring the percentage of coverage of a least cost path from a comparison path (for example, the validation_lcp function).

The index is defined as the area between paths divided by the distance of the shortest path (i.e. Euclidean) between an origin and destination. The index can be interpreted as the average distance between the paths.

`Path Deviation Index = Area between paths / length of shortest path`

The value of the Path Deviation Index depends on the length of the path and makes comparison of PDIs difficult for paths with different origins and destinations. This can be overcome by normalising the Path Deviation Index by the distance of the shortest path (i.e. Euclidean) between an origin and destination.

`Normalised PDI = PDI / length of shortest path x 100`

The normalised Path Deviation Index is the percent of deviation between the two paths over the shortest path. For example, if a normalised PDI is 30 percent, it means that the average distance between two paths is 30 percent of the length of the shortest path. With normalised PDI, all path deviation can be compared regardless of the length of the shortest path.

Note: Direction of lcp and comparison SpatialLines must be in the same order. Check First point (Origin) and Last point (Destination) for confirmation.

Value

SpatialPolygonsDataFrame or SpatialLinesDataFrame (sp package). SpatialPolygonsDataFrame of Area between the LCP and comparison SpatialLines if LCP and comparison SpatialLines are not identical, else returns SpatialLinesDataFrame. Data frame containing Area, PDI, distance of the Euclidean shortest path between the origin and destination and normalised PDI.

Author(s)

Joseph Lewis

References

Jan, O., Horowitz, A.J., Peng, Z.R. (2000). Using Global Positioning System Data to Understand Variations in Path Choice. Transportation Research Record, 1725, 37-44

Examples

```
x1 <- c(1,5,4,50)
y1 <- c(1,3,4,50)
line1 <- sp::SpatialLines(list(sp::Lines(sp::Line(cbind(x1,y1)), ID='a')))
x2 <- c(1,5,5,50)
y2 <- c(1,4,6,50)
line2 <- sp::SpatialLines(list(sp::Lines(sp::Line(cbind(x2,y2)), ID='b')))

val_lcp <- PDI_validation(lcp = line1, line2)
```

---

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

References

Goodchild, F. M., and G. J. Hunter, 1997. A Simple Positional Accuracy Measure for Linear Features. International Journal of Geographical Information Sciences, 11(3), 299-306.

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))
```

---

wide_path_matrix *Create a wide path matrix*

Description

Creates a wide path matrix to be used when calculating wide path least cost paths. This function will return an odd-dimension matrix approximating the shape of an octogon. The centre cell of the matrix has a value of 0 and represents the focal cell. See focal, focalWeight and adjacent for more information.

Usage

```
wide_path_matrix(ncells)
```

Arguments

ncells	numeric value. Dimension of wide path matrix. Note that the value refers to the number of cells and not distance

Value

matrix wide path matrix used when calculating wide path least cost paths via create_wide_lcp

Author(s)

Joseph Lewis

Examples

```
w <- wide_path_matrix(9)
```

Index