Package: movecost (via r-universe)

September 10, 2024

Title Calculation of Slope-Dependant Accumulated Cost Surface, Least-Cost Paths, Least-Cost Corridors, Least-Cost Networks Related to Human Movement Across the Landscape

Version 2.1

Description Provides the facility to calculate non-isotropic accumulated cost surface, least-cost paths, least-cost corridors, least-cost networks using a number of human-movement-related cost functions that can be selected by the user. It just requires a Digital Terrain Model, a start location and (optionally) destination locations. See Alberti (2019) <doi:10.1016/j.softx.2019.100331>.

Depends R (>= 4.0.0)

Imports chron (>= 2.3-56), gdistance (>= 1.2-2), elevatr (>= 0.3.4), Matrix (>= 1.5.0), methods (>= 4.0.3), raster (>= 2.8-4), sf (>= 1.0-9), sp (>= 1.4.0), terra (>= 1.3.0), utils (>= 4.0.0)

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Gianmarco Alberti [aut, cre]

Maintainer Gianmarco Alberti <gianmarcoalberti@gmail.com>

Date/Publication 2024-02-12 18:30:02 UTC

Repository https://gianmarcoalberti.r-universe.dev

RemoteUrl https://github.com/cran/movecost

RemoteRef HEAD

RemoteSha 421e5e240bffd8b3b8e45c0ce90ca6a432c48d2c

Contents

destin.loc . . . 2
Etna_boundary . . . 3
Etna_end_location . . . 3
Etna_start_location . . . 4
malta_dtm_40 . . . 4
movealloc . . . 5
movebound . . . 9
movecomp . . . 13
movecorr . . . 18
movecost . . . 23
movenetw . . . 39
moverank . . . 43
springs . . . 48
volc . . . 48
volc.loc . . . 48

Index **49**

`destin.loc` *Dataset: locations on the volcano Maunga Whau (Auckland, New Zealand)*

Description

A SpatialPointsDataFrame representing spots on the volcano Maunga Whau (Auckland, New Zealand), to be used as destination locations for least-cost paths calculation.

Usage

```
data(destin.loc)
```

Format

SpatialPointsDataFrame

Etna_boundary — *Dataset: bounding polygon representing a study area on Mount Etna (Sicily, Italy)*

Description

A SpatialPolygonDataFrame representing an area on Mount Etna (Sicily, Italy), to be used to download elevation data.

Usage

```
data(Etna_boundary)
```

Format

SpatialPolygonDataFrame

Etna_end_location — *Dataset: locations on Mount Etna (Sicily, Italy)*

Description

A SpatialPointsDataFrame representing spots on Mount Etna (Sicily, Italy), to be used as destination locations for least-cost paths calculation.

Usage

```
data(Etna_end_location)
```

Format

SpatialPointsDataFrame

Etna_start_location *Dataset: location on Mount Etna (Sicily, Italy)*

Description

A SpatialPointsDataFrame representing a spot on Mount Etna, to be used as start location for least-cost paths calculation.

Usage

```
data(Etna_start_location)
```

Format

SpatialPointsDataFrame

malta_dtm_40 *Dataset: Malta DTM (40m cell size)*

Description

A RasterLayer representing a Digital Terrain Model of Malta (40m resolution).

Usage

```
data(malta_dtm_40)
```

Format

RasterLayer

movealloc *R function for calculating slope-dependant walking-cost allocation to origins*

Description

The function provides the facility to carry out a cost allocation analysis. Given a number of origin locations, a cost allocation raster is produced; each cell of the cost allocation raster is given an integer indicating to which origin a cell is closer in terms of cost. Needless to say, the cost can be conceptualized in terms of either walking time or energy expenditure, and is function of the terrain slope.
Visit this LINK to access the package's vignette.

Usage

```
movealloc(
  dtm = NULL,
  origin,
  studyplot = NULL,
  funct = "t",
  time = "h",
  move = 16,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  isolines = FALSE,
  breaks = NULL,
  cont.lab = TRUE,
  cex.breaks = 0.6,
  leg.alloc = FALSE,
  leg.pos = "topright",
  cex.leg = 0.75,
  transp = 0.5,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see movecost).
origin	locations (two at least) in relation to which the cost allocation is carried out (SpatialPointsDataFrame class).

`studyplot` polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see `movecost`); NULL is default.

`funct` cost function to be used (for details on each of the following, see `movecost`):

-functions expressing cost as walking time-
t (default) uses the on-path Tobler's hiking function;
tofp uses the off-path Tobler's hiking function;
mp uses the Marquez-Perez et al.'s modified Tobler's function;
icmonp uses the Irmischer-Clarke's hiking function (male, on-path);
icmoffp uses the Irmischer-Clarke's hiking function (male, off-path);
icfonp uses the Irmischer-Clarke's hiking function (female, on-path);
icfoffp uses the Irmischer-Clarke's hiking function (female, off-path);
ug uses the Uriarte Gonzalez's walking-time cost function;
ma uses the Marin Arroyo's walking-time cost function;
alb uses the Alberti's Tobler hiking function modified for pastoral foraging excursions;
gkrs uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function;
r uses the Rees' hiking function;
ks uses the Kondo-Seino's hiking function;
trp uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;
a uses the Ardigo et al.'s metabolic energy expenditure cost function;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see `movecost`).

`time` time-unit expressed by the isoline(s) if Tobler's and other time-related cost functions are used; h' for hour, 'm' for minutes.

`move` number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).

`cogn.slp` TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see `movecost`).

sl.crit	critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see movecost).
W	walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
L	carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
N	coefficient representing ease of movement (1 by default) (see movecost).
V	speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function (see movecost).
z	zoom level for the elevation data downloaded from online sources (from 0 to 15; 9 by default) (see movecost and get_elev_raster).
isolines	TRUE or FALSE (default) is the user wants or does not want cost isolines/contours around the origins to be calculated and plotted.
breaks	contours' (i.e., isolines') interval; if no value is supplied, the interval is set by default to 1/10 of the range of values of the cost surface accumulated around the origins.
cont.lab	if set to TRUE (default) display the labels of the cost contours.
cex.breaks	set the size of the labels attached to the cost contours (0.6 by default).
leg.alloc	if set to TRUE, display the legend in the plotted cost allocation raster; FALSE by default.
leg.pos	set the position of the legend in the plotted cost allocation raster; 'topright' by default (other options: "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center").
cex.leg	set the size of the labels used in the legend displayed in the plotted allocation raster (0.75 by default).
transp	set the transparency of the slopeshade raster that is plotted over the cost allocation raster (0.5 by default).
export	TRUE or FALSE (default) if the user wants or does not want the output to be exported; if TRUE, the isolines (i.e. the contours) and the allocation boundaries will be exported as a shapefile; the cost allocation raster will be exported as 'GeoTiff'; the DTM is exported only if it was not provided by the user and downloaded by the function from online sources; all the exported files (excluding the DTM) will bear a suffix corresponding to the cost function selected by the user.

Details

The function requires an input DTM ('RasterLayer' class) and a dataset ('SpatialPointsDataFrame' class) containing the origin locations. If a DTM is not provided, `movealloc()` will download elevation data from online sources (see movecost for more details). Under the hood, `movealloc()` relies on the `movecost()` function and implements the same cost functions: see the help documentation of `movecost()` for further information.

Internally, what `movealloc()` does is producing an accumulated cost surface around each individual origin location; those accumulated cost surfaces are then stacked together, and then the function looks at each pixel in the stack of surfaces and returns 1 if the first stacked surface has the smallest pixel value, or 2 if the second stacked surface has the smallest pixel value, and so on for bigger stacks.

`movealloc()` produces a plot featuring a slopeshade image that is overlaid by the cost allocation raster and by a polygon layer where each polygon represents the limits of each allocation zone. A legend can be optionally added to the plot via the `leg.alloc` parameter (FALSE by default). Isolines (i.e., contour lines) around each origin location can be optionally plotted via the 'isolines' parameter (FALSE by default).

The DTM, the cost allocation raster, the cost allocation polygons, and the isolines (if requested by the user by setting the `isolines` parameter to TRUE), can be exported by setting the 'export' parameter to TRUE. All the exported files (excluding the DTM) will bear a suffix corresponding to the cost function selected by the user.

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class)
- cost.allocation.raster: raster of the cost allocation ('RasterLayer' class)
- isolines: contour lines representing the accumulated cost around the origins ('SpatialLinesDataFrame' class); returned if the 'isolines' parameter is set to TRUE
- alloc.boundaries: polygons representing the allocation zones ('SpatialPolygonsDataFrame' class)

See Also

`movecost`

Examples

```
# load a sample Digital Terrain Model
data(volc)


# load the sample locations on the above DTM
data(destin.loc)


#carry out a cost allocation analysis using the Tobler's off-path hiking function,
#setting the time to minutes and the isolines (i.e., contours) interval to 1 minute;
#only 3 locations are used

#result <- movealloc(dtm=volc, origin=destin.loc[c(3,7,9),], funct="tofp", time="m",
#breaks=1, isolines=TRUE)
```

```
#same as above, using all the locations and the Kondo-Seino's cost function

#result <- movealloc(dtm=volc, origin=destin.loc, funct="ks", time="m",
#breaks=1, isolines=TRUE)
```

---

movebound *R function for calculating slope-dependant walking cost boundary(ies) around point location(s)*

Description

The function provides the facility to calculate walking cost boundary(ies) around one or more point locations. Rationale: while `movecost` can calculate and render an accumulated cost surface and corresponding isolines around a point location, the user(s) might want to calculate and plot a boundary (or boundaries) corresponding to a specific walking cost limit around one or more locations, either in terms of walking time or energy expenditure.
Visit this LINK to access the package's vignette.

Usage

```
movebound(
  dtm = NULL,
  origin,
  studyplot = NULL,
  barrier = NULL,
  plot.barrier = FALSE,
  funct = "t",
  time = "h",
  move = 16,
  field = 0,
  cont.value = NULL,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  cont.lab = TRUE,
  transp = 0.5,
  add.geom = FALSE,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see movecost).
origin	location(s) around which the boundary(ies) is calculated (SpatialPointsDataFrame class).
studyplot	polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see movecost); NULL is default.
barrier	area where the movement is inhibited (SpatialLineDataFrame or SpatialPolygonDataFrame class) (see movecost.
plot.barrier	TRUE or FALSE (default) if the user wants or does not want the barrier to be plotted (see movecost).
funct	cost function to be used (for details on each of the following, see movecost):

-functions expressing cost as walking time-
t (default) uses the on-path Tobler's hiking function;
tofp uses the off-path Tobler's hiking function;
mp uses the Marquez-Perez et al.'s modified Tobler's function;
icmonp uses the Irmischer-Clarke's hiking function (male, on-path);
icmoffp uses the Irmischer-Clarke's hiking function (male, off-path);
icfonp uses the Irmischer-Clarke's hiking function (female, on-path);
icfoffp uses the Irmischer-Clarke's hiking function (female, off-path);
ug uses the Uriarte Gonzalez's walking-time cost function;
ma uses the Marin Arroyo's walking-time cost function;
alb uses the Alberti's Tobler hiking function modified for pastoral foraging excursions;
gkrs uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function;
r uses the Rees' hiking function;
ks uses the Kondo-Seino's hiking function;
trp uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;

a uses the Ardigo et al.'s metabolic energy expenditure cost function (for all the mentioned cost functions;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see movecost).

time	time-unit expressed by the isoline(s) if Tobler's and other time-related cost functions are used; h' for hour, 'm' for minutes.
move	number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).
field	value assigned to the cells coincidinng with the barrier (0 by default) (see movecost.
cont.value	cost value represented by the calculated isoline(s) (NULL by default); if no value is supplied, it is set to 1/10 of the range of values of the accumulated cost surface.
cogn.slp	TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see movecost).
sl.crit	critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see movecost).
W	walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
L	carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
N	coefficient representing ease of movement (1 by default) (see movecost).
V	speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function (see movecost).
z	zoom level for the elevation data downloaded from online sources (from 0 to 15; 9 by default) (see movecost and get_elev_raster).
cont.lab	TRUE (default) or FALSE if the usuer wants or does not want labels to be attached to the isolines.
transp	set the transparency of the slopeshade raster that is plotted over the DTM (0.5 by default).
add.geom	TRUE or FALSE (default) if the user wants or does not want the area enclosed by each isolines to be calculated (see Details).
export	TRUE or FALSE (default) if the user wants or does not want the isoline(s) and the copy of the input 'origin' dataset (storing boundaries' geometry information) to be exported; if TRUE, they will be exported as a shapefile; the exported file will bear a suffix corresponding to the cost function selected by the user. The DTM is exported only if it was not provided by the user and downloaded by the function from online sources.

Details

The function just requires an input DTM and a dataset ('SpatialPointsDataFrame' class) containing at least one point location. If a DTM is not provided, movebound() will download elevation

data from online sources (see movecost for more details). Under the hood, movebound() relies on the movecost() function and implements the same cost functions: see the help documentation of movecost() for further information.

The following example uses in-built datasets and calculates 45-minute boundaries around three locations close to Mt Etna (Sicily, Italy), using the Tobler's off-path hiking function (note: elevation data are acquired online for the area enclosed by the polygon fed via the 'studyplot' parameter):

result <- movebound(origin=Etna_end_location, cont.value=45, time="m", cont.lab = TRUE, funct="tofp", studyplot = Etna_boundary, add.geom=TRUE)

Note that by setting the parameter add.geom to TRUE, the function calculates the area enclosed by the boundary represented by each calculated isoline. Needless to say, the unit of measure is the one used by the input layers' coordinate system. The value(s) of the area will be appended as a new variables to a copy of the input 'origin' dataset. The area can only be calculated if the isolines are "complete" and not truncated (i.e., if they do not meet the end of the study area for instance). Therefore, before using this option, the user may want to be sure that all the isolines are actual loops.

With reference to the above example, the area of the three 45-minutes boundaries can be retrieved typing what follows:

result$origin_w_isolines_geom$area

It will return:
17857994 20428575 9172688

that are the values of the area of each 45-minute boundary in square meter.
Needless to say, if we want to convert to square km we can just:
result$origin_w_isolines_geom$area/1000000

which gives
17.857994 20.428575 9.172688

movebound() produces a plot representing the input DTM overlaid by a slopeshade raster, whose transparency can be adjusted using the 'transp' parameter. On the rendered plot, the calculated isoline(s) is displayed and the label(s) representing the cost limit can be activated or deactivated using the 'cont.lab' parameter. The function also returns the isoline(s) ('SpatialLinesDataFrame' class) corresponding to the selected accumulated cost limit and the copy of the 'origin' dataset (storing information about the boundaries' geometry) (see 'Value' below). The isoline(s) and the copy of the 'origin' dataset can be exported as shapefile by setting the export parameter to TRUE.

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class)
- isolines: contour line(s) representing the selected cost limit ('SpatialLinesDataFrame' class)
- origin_w_isolines_geom: copy of the input origin location(s) dataset with a new variable ('area') storing the area values of the boundary calculated around each location

See Also

movecost

Examples

```
# load a sample Digital Terrain Model
data(volc)


# load the sample destination locations on the above DTM
data(destin.loc)


# calculate the 5minute walking time boundary around a location
# using the Tobler's off-path hiking function

result <- movebound(dtm=volc, origin=volc.loc, funct="tofp", move=8, time="m", cont.val=5)


# same as above, but around multiple locations; contours' labels are turned off

result <- movebound(dtm=volc, origin=destin.loc, funct="tofp", move=8, time="m",
cont.val=2, cont.lab=FALSE)
```

---

movecomp *R function for comparing least-cost paths generated using different cost functions*

Description

The function provides the facility to calculate LCPs using different cost functions and to plot them in the same visual output to allow comparability. See, for instance, fig. 14.2 in Parcero-Oubina C. et al, Footprints and Cartwheels on a Pixel Road: On the Applicability of GIS for the Modelling of Ancient (Roman) Routes (2019). In Verhagen P., Joyce J., Groenhuijzen M.R. (eds), Finding the Limits of the Limes. Modelling Demography, Economy and Transport on the Edge of the Roman Empire, Springer, 291-311.
Visit this LINK to access the package's vignette.

Usage

```
movecomp(
  dtm = NULL,
  origin,
  destin,
  studyplot = NULL,
  barrier = NULL,
  plot.barrier = FALSE,
  irregular.dtm = FALSE,
  choice,
  time = "h",
  move = 16,
  field = 0,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  return.base = FALSE,
  leg.pos = "topright",
  leg.cex = 0.75,
  transp = 0.5,
  add.chart = FALSE,
  oneplot = TRUE,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see movecost).
origin	location from which least-cost path(s) is calculated (SpatialPointsDataFrame class).
destin	location(s) to which least-cost path(s) is calculated (SpatialPointsDataFrame class).
studyplot	polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see movecost); NULL is default.
barrier	area where the movement is inhibited (SpatialLineDataFrame or SpatialPolygonDataFrame class) (see movecost).
plot.barrier	TRUE or FALSE (default) if the user wants or does not want the barrier to be plotted (see movecost).
irregular.dtm	TRUE or FALSE (default) if the input DTM features irregular margins (see movecost).

`choice` character vector indicating the cost functions to be compared (for details on each of the following, see `movecost`):

-functions expressing cost as walking time-
t (default) uses the on-path Tobler's hiking function;
tofp uses the off-path Tobler's hiking function;
mp uses the Marquez-Perez et al.'s modified Tobler's function;
icmonp uses the Irmischer-Clarke's hiking function (male, on-path);
icmoffp uses the Irmischer-Clarke's hiking function (male, off-path);
icfonp uses the Irmischer-Clarke's hiking function (female, on-path);
icfoffp uses the Irmischer-Clarke's hiking function (female, off-path);
ug uses the Uriarte Gonzalez's walking-time cost function;
ma uses the Marin Arroyo's walking-time cost function;
alb uses the Alberti's Tobler hiking function modified for pastoral foraging excursions;
gkrs uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function;
r uses the Rees' hiking function;
ks uses the Kondo-Seino's hiking function;
trp uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;
a uses the Ardigo et al.'s metabolic energy expenditure cost function;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see `movecost`).

`time` time-unit to be used if Tobler's and other time-related cost functions are used; h' for hour, 'm' for minutes;

`move` number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).

`field` value assigned to the cells coinciding with the barrier (0 by default) (see `movecost`.

`cogn.slp` TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see `movecost`).

sl.crit	critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see movecost).
W	walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
L	carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
N	coefficient representing ease of movement (1 by default) (see movecost).
V	speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function (see movecost).
z	zoom level for the elevation data downloaded from online sources (from 0 to 15; 9 by default) (see movecost and get_elev_raster).
return.base	TRUE or FALSE (default) if the user wants or does not want the least-cost paths back to the origin to be calculated and plotted (as dashed lines).
leg.pos	set the position of the legend in the plotted cost allocation raster; 'topright' by default (other options: "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center").
leg.cex	set the size of the labels used in the legend displayed in the rendered plot (0.75 by default).
transp	set the transparency of the slopeshade raster that is plotted over the DTM (0.5 by default).
add.chart	TRUE or FALSE (default) is the user wants or does not want boxplots visualising LCPs length/cost vs cost function to be rendered.
oneplot	TRUE (default) or FALSE if the user wants or does not want the plots displayed in a single window.
export	TRUE or FALSE (default) if the user wants or does not want the LCPs to be exported as a shapefile; the DTM is exported only if it was not provided by the user and downloaded by the function from online sources.

Details

Like movecost(), the function just requires an input DTM ('RasterLayer' class), and an origin and destination dataset ('SpatialPointsDataFrame' class). The cost functions to be used have to be entered into 'movecomp()' via a character vector fed via the choice parameter (see the examples below). If a DTM is not provided, movecomp() downloads elevation data from online sources for the area enclosed by the polygon fed via the studyplot parameter (see movecost for more details). Under the hood, movecomp() relies on movecost() and implements the same cost functions: see the help documentation of movecost() for further information.

movecomp() produces a plot representing the input DTM overlaid by a slopeshade raster, whose transparency can be adjusted using the 'transp' parameter. On the rendered plot, the LPCs ('SpatialLinesDataFrame' class) generated by the different input cost functions are given a different line type; a legend indicates which line type corresponds to which cost function. LCPs back to the origin

can be calculated (and plotted) setting the parameter `return.base` to `TRUE`.

The function returns the LCPs and (if requested by the user) the LCPs back to the origin. If the DTM has been acquired online, it will be returned as well. The LCPs (and the LCPs back to the origin) will store three variables: the length of each path, the cost of each path, and an abbreviation corresponding to the cost function used to generate the LCPs. The mentioned data can be exported by setting the `export` parameter to `TRUE`.

If the users want to compare the distribution of the length of the LCPs generate by different cost functions, it suffices to set the `add.chart` parameter to `TRUE`. Two charts featuring boxplots will be rendered: one plotting the distribution of the LCPs length by cost function; one portaying the distribution of the cost by cost function.

The following example uses in-built datasets to compare the LCPs generated using two cost functions: the Tobler hiking function , the wheeled vehicle cost function, and the Pantolf. et al's cost function with correction factor. LCPs back to the origin location will be calculated as well. The origin and destination locations are close to Mt Etna (Sicily, Italy). Note that elevation data are acquired online for the area enclosed by the polygon fed via the `studyplot` parameter:

result <- movecomp(origin=Etna_start_location, destin=Etna_end_location, choice=c("t", "wcs", "pcf"), studyplot = Etna_boundary, return.base=TRUE)

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class); returned only if acquired online
- LCPs: estimated least-cost paths ('SpatialLinesDataFrame' class); three variables are stored: 'length', 'cost', and 'funct'.
- LCPs.back: estimated least-cost paths back to the origin ('SpatialLinesDataFrame' class); three variables are stored; see above.

See Also

movecost

Examples

```
# load a sample Digital Terrain Model
data(volc)


# load the sample destination locations on the above DTM
data(destin.loc)


# compare the LCPs generated using different walking-time cost functions (time in minutes)
```

```
result <- movecomp(volc, volc.loc, destin.loc, choice=c("t", "ug", "gkrs"), time="m", move=8)

# the distribution of the length and cost of the LCPs by cost function can be easily compared
# using the 'add.chart' parameter:
#result <- movecomp(volc, volc.loc, destin.loc, choice=c("t", "ug", "gkrs"), time="m",
#move=8, add.chart=T)
```

---

movecorr *R function for calculating least-cost corridor between point locations*

Description

The function provides the facility to calculate the least-cost corridor between point locations. It just requires an input DTM and at least two point locations ('SpatialPointsDataFrame' class) representing the locations between which the corridor is calculated. Under the hood, `movecorr()` relies on the `movecost` function and, needless to say, implements the same cost functions. See the help documentation of 'movecost()' for further details.
Visit this LINK to access the package's vignette.

Usage

```
movecorr(
  dtm = NULL,
  a,
  b,
  lab.a = "A",
  lab.b = "B",
  cex.labs = 0.8,
  studyplot = NULL,
  barrier = NULL,
  plot.barrier = FALSE,
  irregular.dtm = FALSE,
  funct = "t",
  time = "h",
  move = 16,
  field = 0,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  rescale = FALSE,
```

```
  transp = 0.5,
  graph.out = TRUE,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see movecost).
a	first location from which the least-cost corridor is calculated (SpatialPointsDataFrame class); if it contains more than two locations, see the 'Description' section above.
b	second location from which the least-cost corridor is calculated (SpatialPointsDataFrame class); if parameter 'a' stores more than two locations, this parameter is disregarded; see the 'Description' section above.
lab.a	string to be used to label point a on the outplut plot (A is the default)
lab.b	string to be used to label point a on the outplut plot (B is the default).
cex.labs	scaling factor for the size of the points' labels (0.8 by default)
studyplot	polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see movecost); NULL is default.
barrier	area where the movement is inhibited (SpatialLineDataFrame or SpatialPolygonDataFrame class) (see movecost).
plot.barrier	TRUE or FALSE (default) if the user wants or does not want the barrier to be plotted (see movecost).
irregular.dtm	TRUE or FALSE (default) if the input DTM features irregular margins (see movecost).
funct	cost function to be used (for details on each of the following, see movecost):

-functions expressing cost as walking time-
t (default) uses the on-path Tobler's hiking function;
tofp uses the off-path Tobler's hiking function;
mp uses the Marquez-Perez et al.'s modified Tobler's function;
icmonp uses the Irmischer-Clarke's hiking function (male, on-path);
icmoffp uses the Irmischer-Clarke's hiking function (male, off-path);
icfonp uses the Irmischer-Clarke's hiking function (female, on-path);
icfoffp uses the Irmischer-Clarke's hiking function (female, off-path);
ug uses the Uriarte Gonzalez's walking-time cost function;
ma uses the Marin Arroyo's walking-time cost function;
alb uses the Alberti's Tobler hiking function modified for pastoral foraging excursions;
gkrs uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function;
r uses the Rees' hiking function;
ks uses the Kondo-Seino's hiking function;
trp uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;
a uses the Ardigo et al.'s metabolic energy expenditure cost function;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see movecost).

`time` time-unit expressed by the accumulated raster if Tobler's and other time-related cost functions are used; h' for hour, 'm' for minutes.

`move` number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).

`field` value assigned to the cells coinciding with the barrier (0 by default) (see movecost.

`cogn.slp` TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see movecost).

`sl.crit` critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see movecost).

`W` walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).

`L` carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).

`N` coefficient representing ease of movement (1 by default) (see movecost).

`V` speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function (see movecost).

`z` zoom level for the elevation data downloaded from online sources (from 0 to 15; 9 by default) (see movecost and get_elev_raster).

`rescale` TRUE or FALSE (default) if the user wants or does not want the output least-coast corridor raster to be rescaled between 0 and 1.

`transp` set the transparency of the slopeshade raster that is plotted over the least-cost corridor raster (0.5 by default).

`graph.out` TRUE (default) or FALSE if the user wants or does not want a graphical output to be generated.

`export` TRUE or FALSE (default) if the user wants or does not want the output to be exported; if TRUE, the least-cost corridor, the dtm (if not provided by the user but acquired online), and the accumulated cost surface around a and b are exported as a GeoTiff file, while the two LCPs (from a to b, and from b to a) as individual shapefiles. If multiple locations are analysed, only the least-cost corridor (and the DTM if originally not provided) will be exported. All the exported files (excluding the DTM) will bear a suffix corresponding to the cost function selected by the user.

Details

If only two locations are provided (one via parameter `a`, one via parameter `b`), the function renders a raster representing the least cost corridor (which can be optionally exported as GeoTiff) with least-cost paths superimposed. If more than 2 locations are fed into the function via the 'a' parameter, the function calculates the least-cost corridor between pairs of locations. All the pair-wise corridor rasters are returned (but not individually plotted) in a list. All those rasters will be summed, and the resulting raster will be plotted (and can be, optionally, exported as GeoTiff).

The function returns a list containing a number of components (see 'Value' below). For more details about exporting the function's outputs, see 'Arguments' below.

If the user wants to calculate the least-cost corridor between two locations only, (s)he may want to use parameter `a` and `b` to indicate the two locations of interest respectively. For example, using the datasets provided by this package:

result <- movecorr(a=Etna_start_location, b=Etna_end_location[1,], studyplot=Etna_boundary, funct="tofp")

The above will produce the least-cost corridor between two locations close to Mt Etna (Sicily, Italy), using the Tobler's cost function (for off-path hiking). Side note: the elevation data will be acquired online.

If the interest lies in using more than 2 locations, the user may want to feed the dataset storing all the locations into parameter `a` (disregarding `b`). As explained above, in this case the function calculates the least-cost corridor between pairs of locations. All the pair-wise corridor rasters are returned in a list. Those rasters will be summed, and the resulting raster will be plotted (and can be, optionally, exported as GeoTiff). For example, to calculate the least-cost corridors between every individual unique pair of the 9 locations stored in the `destin.loc` dataset:

volc <- raster::raster(system.file("external/maungawhau.grd", package="gdistance"))

result <- movecorr(dtm=volc, a=destin.loc, funct="ree", rescale=TRUE)

Note that only parameter `a` has been used. The function returns and plots the sum of the 36 individual corridors; the latter are not plotted, but are stored in a list. If the user wants to plot the least-cost

corridor, say, n 4, and then add the two locations between which the corridor has been calculated, (s)he can first plot the corridor raster n 4:

raster::plot(result$corridors[[4]])

Then, identifying which locations are related to corridor n 4 can be easily accomplished by looking up the values stored in the 4th column of the returned matrix:

result$locations.matrix

The locations are the n 1 and n 5, so the user can add them to the plot previosly produced using:

raster::plot(destin.loc[1,], pch=20, add=T)
raster::plot(destin.loc[5,], pch=20, add=T)

Note that the resulting plot can be produced (with a nicer outlook) directly by 'movecorr()' by feeding those two locations in the parameter 'a' and 'b' respectively:

result <- movecorr(dtm=volc, a=destin.loc[1,], b=destin.loc[5,], funct="ree")

Overall, what `movecorr()` does is to calculate (via the `movecost` function) the accumulated cost surface around each location. Those are eventually summed to produce the least-cost corridor between locations. On the produced corridor raster, the cost of a cell is the total cost to reach it from all the analysed locations. About least-cost corridors between pairs of locations, see for instance: Mitchell A. (2012), "The ESRI Guide to GIS Analysis. Vol 3. Modelling Suitability, Movement, and Interaction", New York: Esri Press (257-259).

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class)
- lc.corridor: raster of the least-cost corridor ('RasterLayer' class); if more than two locations are analysed, this raster is the sum of all the corridors between all the pairs of locations
- lcp_a_to_b: least-cost past from a to b ('SpatialLinesDataFrame' class); returned only when the corridor is calculated between two locations
- lcp_b_to_a: least-cost past from b to a ('SpatialLinesDataFrame' class); returned only when the corridor is calculated between two locations
- accum_cost_surf_a: accumulated cost-surface around a ('RasterLayer' class); returned only when the corridor is calculated between two locations
- accum_cost_surf_b: accumulated cost-surface around b ('RasterLayer' class); returned only when the corridor is calculated between two locations
- corridors: list of rasters ('RasterLayer' class) representing the least-cost corridor between all the unique pairs of locations; returned only when more than two locations are analysed

- locations.matrix: matrix whose columns indicate the identifiers for all the unique pairs of locations for which each corridor is calculated; returned only when more than two locations are analysed

See Also

movecost

Examples

```
# load a sample Digital Terrain Model
data(volc)


# load the sample destination locations on the above DTM
data(destin.loc)


# calculate the least-cost corridor between two locations, using the
# relative energetic expenditure cost function, and store the results
# in the 'result' object

result <- movecorr(dtm=volc, a=destin.loc[1,], b=destin.loc[3,], funct="ree", move=8)


#same as above, but using the 'cognitive slope'

# result <- movecorr(dtm=volc, a=destin.loc[1,], b=destin.loc[3,],
# funct="ree", move=8, cogn.slp=TRUE)
```

`movecost` *R function for calculating accumulated anisotropic slope-dependant cost of movement across the terrain and least-cost paths from a point origin*

Description

The function provides the facility to calculate the anisotropic accumulated cost of movement around a starting location and to optionally calculate least-cost path(s) toward one or multiple destinations. It implements different cost estimations related to human movement across the landscape. The function takes as input a Digital Terrain Model ('RasterLayer' class) and a point feature ('SpatialPointsDataFrame' class), the latter representing the starting location, i.e. the location from which the accumulated cost is calculated. Besides citing this package, you may want to refer to the following journal article, where an earlier version of the package is described: **Alberti (2019) <doi:10.1016/j.softx.2019.100331>**.
Visit this LINK to access the package's vignette.

Usage

```
movecost(
  dtm = NULL,
  origin,
  destin = NULL,
  studyplot = NULL,
  barrier = NULL,
  plot.barrier = FALSE,
  irregular.dtm = FALSE,
  funct = "t",
  time = "h",
  outp = "r",
  move = 16,
  field = 0,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  return.base = FALSE,
  rb.lty = 2,
  breaks = NULL,
  cont.lab = TRUE,
  destin.lab = TRUE,
  cex.breaks = 0.6,
  cex.lcp.lab = 0.6,
  graph.out = TRUE,
  transp = 0.5,
  oneplot = TRUE,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see Details).
origin	location from which the cost surface is calculated (SpatialPointsDataFrame class).
destin	location(s) to which least-cost path(s) is calculated (SpatialPointsDataFrame class).
studyplot	polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see Details); NULL is default.
barrier	area where the movement is inhibited (SpatialLineDataFrame or SpatialPolygonDataFrame class).
plot.barrier	TRUE or FALSE (default) if the user wants or does not want the barrier to be plotted (in blue).

`irregular.dtm` TRUE or FALSE (default) if the input DTM features irregular margins (Details).

`funct` cost function to be used:

-functions expressing cost as walking time-
t (default) uses the on-path Tobler's hiking function;
tofp uses the off-path Tobler's hiking function;
mp uses the Marquez-Perez et al.'s modified Tobler's function;
icmonp uses the Irmischer-Clarke's hiking function (male, on-path);
icmoffp uses the Irmischer-Clarke's hiking function (male, off-path);
icfonp uses the Irmischer-Clarke's hiking function (female, on-path);
icfoffp uses the Irmischer-Clarke's hiking function (female, off-path);
ug uses the Uriarte Gonzalez's walking-time cost function;
ma uses the Marin Arroyo's walking-time cost function;
alb uses the Alberti's Tobler hiking function modified for pastoral foraging excursions;
gkrs uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function;
r uses the Rees' hiking function;
ks uses the Kondo-Seino's hiking function;
trp uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;
a uses the Ardigo et al.'s metabolic energy expenditure cost function;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see Details).

`time` time-unit expressed by the accumulated raster and by the isolines if Tobler's and other time-related cost functions are used; 'h' for hour, 'm' for minutes.

`outp` type of output: 'raster' or 'contours' (see Details).

`move` number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).

`field` value assigned to the cells coinciding with the barrier (0 by default).

cogn.slp	TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see Details).
sl.crit	critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see Details).
W	walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see Details).
L	carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see Details).
N	coefficient representing ease of movement (1 by default) (see Details).
V	speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function; see Details).
z	zoom level for the elevation data downloaded from online sources (0 to 15; 9 by default) (see Details and get_elev_raster).
return.base	TRUE or FALSE (default) if the user wants or does not want the least-cost paths back to the origin to be calculated and plotted (as dashed lines).
rb.lty	line type used to represent the least-cost paths back to the origin in the returned plot (2 by default; dashed line; see 'lty' parameter in par).
breaks	contour interval; if no value is supplied, the interval is set by default to 1/10 of the range of values of the accumulated cost surface.
cont.lab	if set to TRUE (default) display the labels of the contours over the accumulated cost surface.
destin.lab	if set to TRUE (default) display the label(s) indicating the cost at the destination location(s).
cex.breaks	set the size of the cost labels used in the contour plot (0.6 by default).
cex.lcp.lab	set the size of the labels used in least-cost path(s) plot (0.6 by default).
graph.out	TRUE (default) or FALSE if the user wants or does not want a graphical output to be generated.
transp	set the transparency of the slopeshade raster that is plotted over the rendered plots (0.5 by default).
oneplot	TRUE (default) or FALSE if the user wants or does not want the plots displayed in a single window.
export	TRUE or FALSE (default) if the user wants or does not want the outputs to be exported; if TRUE, the DTM, the cost-surface, and the accumulated cost surface are exported as a GeoTiff file, while the isolines, the least-cost path(s), and a copy of the input destination locations (storing the cost measured at each location) are exported as shapefile; all the exported files (excluding the DTM) will bear a suffix corresponding to the cost function selected by the user. Note that the DTM is exported only if it was not provided by the user and downloaded by the function from online sources.

Details

If the parameter `destin` is fed with a dataset representing destination location(s) ('SpatialPointsDataFrame' class), the function also calculates least-cost path(s) plotted on the input DTM; the length of each path is saved under the variable 'length' stored in the 'LCPs' dataset ('SpatialLines' class) returned by the function. In the produced plot, the red dot(s) representing the destination location(s) are labelled with numeric values representing the cost value at the location(s).

The cost value is also appended to the updated destination locations dataset returned by the function, which stores a new variable labelled `cost`. If the cost is expressed in terms of walking time, the labels accompaining each destinaton location will express time in sexagesimal numbers (hours, minutes, seconds). In this case, the variable 'cost' appended to the returned destination location datset will store the time figures in decimal numbers, while another variable named `cost_hms` will store the corresponding value in sexagesimal numbers. When interpreting the time values stored in the `cost` variable, the user may want to bear in mind the selected time unit (see right below).

When using cost functions expressing cost in terms of time, the time unit can be selected by the user setting the `time` parameter to `h` (hours) or to `m` (minutes).

In general, the user can also select which type of visualization the function has to produce; this is achieved setting the `outp` parameter to either `r` (=raster) or to `c` (=contours). The former will produce a raster with a colour scale and contour lines representing the accumulated cost surface; the latter parameter will only produce contour lines.

The contour lines' interval is set using the `breaks` parameter; if no value is passed to the parameter, the interval will be set by default to 1/10 of the range of values of the accumulated cost surface.

It is worth reminding the user(s) that all the input layers (i.e., DTM, start location, and destination locations) must use the same projected coordinate system.

Cost surface calculation:
for the cost-surface and LCPs calculation, `movecost()` builds on functions from Jacob van Etten's gdistance package. Under the hood, `movecost()` calculates the slope as rise over run, following the procedure described by van Etten, "R Package gdistance: Distances and Routes on Geographical Grids" in Journal of Statistical Software 76(13), 2017, pp. 14-15. The number of directions in which cells are connected in the cost calculation can be set to 4 (rook's case), 8 (queen's case), or 16 (knight and one-cell queen moves) using the `move` parameter (see 'Arguments').

Inhibition of movement (barrier):
areas where the movement is inhibited can be fed into the analysis via the `barrier` parameter; SpatialLineDataFrame or SpatialPolygonDataFrame can be used. The barrier is assigned a conductance value of 0 (i.e., movement is inhibited) by default, but the user can assign any other value via the `field` parameter. Internally, the barrier creation rests on the internal `create_barrier_cs` function, which relies on the same function out of an earlier version of the *leastcostpath* package.

To test this facility, consider the following example:

First, we use in-built data to come up with a linear feature (i.e., a LCP) that we will later use as barrier:

result1 <- movecost(volc, destin.loc[1,], destin.loc[4,])

After, we calculate the LCP between two other locations, first not using any barrier (result2), then using the mentioned LCP (from result1) as a barrier (result3):

result2 <- movecost(volc, destin.loc[3,], destin.loc[6,], move=8)
result3 <- movecost(volc, destin.loc[3,], destin.loc[6,], barrier=result1$LCPs, plot.barrier=TRUE, move=8)

As apparent by comparing result2 to result3, when the barrier is used (result3), the LCP does not cross the barrier but is "forced" to make a long detour. In result3, the barrier is plotted as a blue line. **Note** that the `move` parameter has been set to 8; if set to 16, the LCP will be "able" to jump the barrier.

DTM featuring irregular margins:
if the input DTM features irregular margins, e.g a coastline with gulfs and/or inlets where cells corresponding to the sea are given NoData, the user is to set the `irregular.dtm` parameter to TRUE; this will prevent the LCPs to cross the sea. Internally, what `movecost()` does is to generate a polygon vector layer from the DTM and to use the polygon as a mask to create a Transitional Layer via the internal `create_barrier_cs` function, which relies on the same function out of an earlier version from the *leastcostpath* package. In the mask Transitional Layer those parts corresponding to the terrain are given a conductance value equal to 1, while everything else (i.e., the parts corresponding to the sea) are given 0 conductance. The mask Transitional Layer is then internally multiplied by the conductance transitional layer representing the cost of movement (according to the user-selected function). This will set to 0 the conductance values of those parts of the study area that do not correspond to the terrain, while keeping unaltered the conductance of those parts that do coincide with the terrain.

As a case in point, let's consider the two following examples (using some in-build datasets):

resultA <- movecost(malta_dtm_40, origin=springs[5,], destin=springs[15,], irregular.dtm=FALSE, oneplot=FALSE)

resultB <- movecost(malta_dtm_40, origin=springs[5,], destin=springs[15,], irregular.dtm=TRUE, oneplot=FALSE)

As you can see, in the first case, the LCP between the two locations cross the sea, while in the second case the LCP follows the coastline. One can also appreciate the difference between the two returned conductance transitional layers:

raster::plot(raster::raster(resultA$conductance))

raster::plot(raster::raster(resultB$conductance))

It is apparent that in the second layer the sea area has been given 0 conductance, while keeping the rest unchanged. If the input DTM does not feature irregular margins (like, for instance, the built-in `volc` DTM), the user may safely leave the `irregular.dtm` parameter set to FALSE (which is the default value).

Acquiring online elevation data:
if a DTM is not provided,`movecost()`' will download elevation data from online sources. Elevation data will be acquired for the area enclosed by the polygon supplied by the `studyplot` parameter (SpatialPolygonDataFrame class). To tap online elevation data, `movecost()`' internally builds on the `get_elev_raster` function from the *elevatr* package.

The zoom level of the downloaded DTM (i.e., its resolution) is controlled by the parameter `z`, which is set to 9 by default (a trade off between resolution and download time).

To test this facility, the user may want to try the following code, that will generate a least-cost surface and least-cost paths in an area close the Mount Etna (Sicily, Italy), whose elevation data are acquired online; the start and end locations, and the polygon defining the study area, are provided in this same package:

result <- movecost(origin=Etna_start_location, destin=Etna_end_location, studyplot=Etna_boundary)

The LCPs back to the origin can be calculated and plotted setting the parameter 'return.base' to TRUE:

result <- movecost(origin=Etna_start_location, destin=Etna_end_location, studyplot=Etna_boundary, return.base=TRUE)

To know more about what elevation data are tapped from online sources, visit: https://cran.r-project.org/web/packages/elevatr/vignettes/introduction_to_elevatr.html.

For more information about the elevation data resolution per zoom level, visit https://github.com/tilezen/joerd/blob/master/doc
sources.md#what-is-the-ground-resolution.

To know what is sourced at what zoom level, visit https://github.com/tilezen/joerd/blob/master/docs/data-sources.md#what-is-sourced-at-what-zooms.

Terrain slope and cognitive slope:
when it comes to the terrain slope, the function provides the facility to use the so-called 'cognitive slope', following Pingel TJ (2013), Modeling Slope as a Contributor to Route Selection in Mountainous Areas, in Cartography and Geographic Information Science, 37(2), 137-148. According to Pingel, "Humans tend to overestimate geographic slopes by a surprisingly high margin...This analysis indicates downhill slopes are overestimated at approximately 2.3 times the vertical, while uphill slopes are overestimated at 2 times the vertical.". As a result, if the parameter `cogn.slp` is

set to TRUE, positive slope values are preliminarily multiplied by 1.99, while negative slope values are multiplied by 2.31.

Terrain factor (N):
virtually all the implemented cost functions (with few exceptions) can take into account a 'terrain factor' (N parameter; 1 by default), which represents the easiness/difficulty of moving on different terrain types. According to the type of terrain, the energy spent when walking increases. The same holds true for time, which increases because on a loose terrain (for instance) the walking speed decreases. While a terrain factor is 'natively' part of the Van Leusen's, Pandolf et al.'s, and Bellavia's cost function, it has been integrated into the other cost functions as well (when/if relevant).

Note that the terrain factor has NOT been implemented in the Alberti's, Tobler's off-path, and Irmischer-Clarke's off-path cost function. As for the latter two, they already natively feature a terrain factor. Therefore, it has been implemented only in their on-path version. Needless to say, if we use a terrain factor of 1.67 with the Tobler's (on-path) hiking function, the results will be equal to those obtained using the Tobler's off-path function (the reciprocal of 1.67, i.e. 0.60, is in fact natively used by the Tobler's function for off-path hiking). In fact, compare the results of the following two runs of `movecost()` (using in-built datasets):

```
result1 <- movecost(dtm=volc, origin=volc.loc, destin=destin.loc, breaks=0.05, funct="t", N=1.67)
result2 <- movecost(dtm=volc, origin=volc.loc, destin=destin.loc, breaks=0.05, funct="tofp")
```

The user may want to refer to the following **list of terrain factors**, which is based on the data collected in Herzog, I. (2020). Spatial Analysis Based on Cost Functions. In Gillings M, Haciguzeller P, Lock G (eds), "Archaeological Spatial Analysis. A Methodological Guide.", Routledge: New York, 340 (with previous references). The list is divided into two sections (a and b), the first reporting the terrain factors to be used for cost functions measuring time, the second for functions measuring cost other than time:

(a)

- Blacktop roads, improved dirt paths, cement = 1.00
- Lawn grass = 1.03
- Loose beach sand = 1.19
- Disturbed ground (former stone quarry) = 1.24
- Horse riding path, flat trails and meadows = 1.25
- Tall grassland (with thistle and nettles) = 1.35
- Open space above the treeline (i.e., 2000 m asl) = 1.50
- Bad trails, stony outcrops and river beds = 1.67
- Off-paths = 1.67
- Bog = 1.79
- Off-path areas below the treeline (pastures, forests, heathland) = 2.00
- Rock = 2.50
- Swamp, water course = 5.00

(b)

- Asphalt/blacktop = 1.00
- Dirt road or grass = 1.10
- Hard-surface road = 1.20
- Light brush = 1.20
- Ploughed field = 1.30 or 1.50
- Heavy brush = 1.50
- Hard-packed snow = 1.60
- Swampy bog = 1.80
- Sand dunes = 1.80
- Loose sand = 2.10

Implemented cost functions:
note that in what follows **x[adj]** stands for slope as rise/run calculated for adjacent cells:

Tobler's hiking function (on-path) (speed in kmh):

$$(6 * exp(-3.5 * abs(x[adj] + 0.05))) * (1/N)$$

Tobler's hiking function (off-path) (speed in kmh):

$$(6 * exp(-3.5 * abs(x[adj] + 0.05))) * 0.6$$

as per Tobler's indication, the off-path walking speed is reduced by 0.6.

Marquez-Perez et al.'s modified Tobler hiking function (speed in kmh):

$$(4.8 * exp(-5.3 * abs((x[adj] * 0.7) + 0.03))) * (1/N)$$

modified version of the Tobler's hiking function as proposed by Joaquin Marquez-Parez, Ismael Vallejo-Villalta & Jose I. Alvarez-Francoso (2017), "Estimated travel time for walking trails in natural areas", Geografisk Tidsskrift-Danish Journal of Geography, 117:1, 53-62, DOI: 10.1080/00167223.2017.1316212.

Irmischer-Clarke's modified Tobler hiking function (male, on-path; speed in kmh):

$$((0.11 + exp(-(abs(x[adj]) * 100 + 5)^2/(2 * 30^2))) * 3.6) * (1/N)$$

modified version of the Tobler's function as proposed for (male) on-path hiking by Irmischer, I. J., & Clarke, K. C. (2018). Measuring and modeling the speed of human navigation. Cartography and Geographic Information Science, 45(2), 177-186. https://doi.org/10.1080/15230406.2017.1292150. It is interesting to note that the hiking speed predicted by this and by the other functions proposed

by the authors is slower than the one modelled by Tobler's hiking function. This is attributed to the cognition involved in wayfinding , such as map reading, analyzing the terrain, decision making, determining routes, etc. **Note**: all the all the Irmischer-Clarke's functions originally express speed in m/s; they have been reshaped (multiplied by 3.6) to turn m/s into km/h for consistency with the other Tobler-related cost functions; slope is in percent.

Irmischer-Clarke's modified Tobler hiking function (male, off-path; speed in kmh):

$(0.11 + 0.67 * exp(-(abs(x[adj]) * 100 + 2)^2/(2 * 30)^2)) * 3.6$

Irmischer-Clarke's modified Tobler hiking function (female, on-path; speed in kmh):

$((0.95 * (0.11 + exp(-(abs(x[adj]) * 100 + 5)^2/(2 * 30^2)))) * 3.6) * (1/N)$

Irmischer-Clarke's modified Tobler hiking function (female, off-path; speed in kmh):

$(0.95 * (0.11 + 0.67 * exp(-(abs(x[adj]) * 100 + 2)^2/(2 * 30^2)))) * 3.6$

Uriarte Gonzalez's walking-time cost function:

$1/((0.0277 * (abs(x[adj]) * 100) + 0.6115) * N)$

proposed by Uriarte Gonzalez; **see**: Chapa Brunet, T., Garcia, J., Mayoral Herrera, V., & Uriarte Gonzalez, A. (2008). GIS landscape models for the study of preindustrial settlement patterns in Mediterranean areas. In Geoinformation Technologies for Geo-Cultural Landscapes (pp. 255-273). CRC Press. https://doi.org/10.1201/9780203881613.ch12.
The cost function originally expresses walking time in seconds; for the purpose of its implementation in this function, it is the reciprocal of time (1/T) that is used in order to eventually get T/1. Unlike in the original cost function, here the pixel resolution is not taken into account since 'gdistance' takes care of the cells' dimension when calculating accumulated costs.

Marin Arroyo's walking-time cost function:

$ifelse((abs(x[adj])*100) < 0, 1/((0.6*((abs(x[adj])*100)/23+1))*N), 1/((0.6*((abs(x[adj])*100)/11 + 1)) * N))$

used by Marin Arroyo A.B. (2009), The use of optimal foraging theory to estimate Late Glacial site catchments areas from a central place: the case of eastern Cantabria, Span, in Journal of Anthropological Archaeology 28, 27-36. The cost function originally expresses walking time in seconds; here it is the reciprocal of time (1/T) that is used in order to eventually get T/1. Slope is in percent. Note: unlike in the original equation, here d (distance travelled in meter) is not taken into account since 'gdistance' takes care of the cells' dimension when calculating accumulated costs.

Alberti's Tobler hiking function modified for pastoral foraging excursions (speed in kmh):

$(6 * exp(-3.5 * abs(x[adj] + 0.05))) * 0.25$

proposed by Gianmarco Alberti; **see**: Locating potential pastoral foraging routes in Malta through the use of a Geographic Information System. The Tobler's function has been rescaled to fit animal walking speed during foraging excursions. The distribution of the latter, as empirical data show, turns out to be right-skewed and to vary along a continuum. It ranges from very low speed values (corresponding to slow grazing activities grazing while walking) to comparatively higher values (up to about 4.0 km/h) corresponding to travels without grazing (directional travel toward feeding stations). The function consider 1.5 km/h as the average flock speed, which roughly corresponds to the average speed recorded in some studies, and can be considered the typical speed of flocks during excursions in which grazing takes place while walking (typical form of grazing in most situations). Tobler's hiking function has been rescaled by a factor of 0.25 to represent the walking pace of a flock instead of humans. The factor corresponds to the ratio between the flock average speed (1.5 km/h) and the maximum human walking speed (about 6.0 km/h) on a favourable slope.

Garmy, Kaddouri, Rozenblat, and Schneider's hiking function (speed in kmh):

$(4 * exp(-0.008 * ((atan(abs(x[adj])) * 180/pi)^2))) * (1/N)$

slope in degrees; **see**: Herzog, I. (2020). Spatial Analysis Based on Cost Functions. In Gillings M, Haciguzeller P, Lock G (eds), "Archaeological Spatial Analysis. A Methodological Guide.", Routledge: New York, 333-358 (with previous references).

Rees' hiking function (speed in kmh):

$((1/(0.75 + 0.09 * abs(x[adj]) + 14.6 * (abs(x[adj]))^2)) * 3.6) * (1/N)$

Rees' slope-dependant cost function; it is originally expressed in terms of time (1/v in Rees' publication); here it is the reciprocal of time (i.e. speed) that is used in order to eventually get the reciprocal of speed (i.e. time). Slope is dealt with here as originally expressed in Rees' publication (i.e. rise over run). The speed, which is originally expressed in m/s, has been here transposed to kmh (i.e., multiplied by 3.6) for consistency with other hiking functions.
For this cost function **see**: Rees, WG (2004). Least-cost paths in mountainous terrain. Computers & Geosciences, 30(3), 203-209. See also: Campbell MJ, Dennison PE, Butler BW, Page WG (2019). Using crowdsourced fitness tracker data to model the relationship between slope and travel rates. Applied Geography 106, 93-107 (with previous references).

Kondo-Seino's modified Tobler hiking function (speed in kmh):

$ifelse(abs(x[adj]) >= -0.07, (5.1 * exp(-2.25 * abs(x[adj] + 0.07))) * (1/N), (5.1 * exp(-1.5 * abs(x[adj] + 0.07)))) * (1/N)$

Kondo-Seino's modified Tobler hiking function; it expresses walking speed in Kmh; slope as rise/run; **see** Kondo Y., Seino Y. (2010). GPS-aided Walking Experiments and Data-driven Travel Cost Modelingon the Historical Road of Nakasendo-Kisoji (Central Highland Japan), in: Frischer

B., Webb Crawford J., Koller D. (eds.), Making History Interactive. Computer Applications and Quantitative Methods in Archaeology (CAA). Proceedings of the 37th International Conference, Williamsburg, Virginia, United States of America, March 22-26 (BAR International Series S2079). Archaeopress, Oxford, 158-165.

Tripcevich hiking function (speed in kmh):

$((4.028 * 46^2)/(((atan(abs(x[adj])) * 180/pi) + 4.127)^2 + 46^2)) * (1/N)$

Tripcevich's hiking function; it expresses walking speed in Kmh; slope is originally expressed in degrees; **see** Tripcevich N (2008). Estimating Llama caravan travel speeds: ethno-archaeological fieldwork with a Peruvian salt caravan. Trabajo presentado el la inauguracionn del Centre for Spatial Studies, University of California, Santa Barbara. See also: Lucero G, Marsh EJ, Castro S (2014), Rutas prehistoricas en lo NO de San Juan: una propuesta macroregional desde los sistemas de information geografica, in Cortegoso V, Duran V, Gasco Alejandra (eds), Arqueologia de ambientes de altura de Mendoza y San Juan (Argentina), EDIUNC, pp. 275-305.

Wheeled-vehicle critical slope cost function:

$1/((1 + ((abs(x[adj]) * 100)/sl.crit)^2) * N)$

where $sl.crit$ (=critical slope, in percent) is "the transition where switchbacks become more effective than direct uphill or downhill paths" and typically is in the range 8-16; **see** Herzog, I. (2016). Potential and Limits of Optimal Path Analysis. In A. Bevan & M. Lake (Eds.), Computational Approaches to Archaeological Spaces (pp. 179-211). New York: Routledge.

Relative energetic expenditure cost function:

$1/((tan((atan(abs(x[adj])) * 180/pi) * pi/180)/tan(1 * pi/180)) * N)$

slope-based cost function expressing change in potential energy expenditure; **see** Conolly, J., & Lake, M. (2006). Geographic Information Systems in Archaeology. Cambridge: Cambridge University Press, p. 220; **see also** Newhard, J. M. L., Levine, N. S., & Phebus, A. D. (2014). The development of integrated terrestrial and marine pathways in the Argo-Saronic region, Greece. Cartography and Geographic Information Science, 41(4), 379-390, with references to studies that use this function; **see also** ten Bruggencate, R. E., Stup, J. P., Milne, S. B., Stenton, D. R., Park, R. W., & Fayek, M. (2016). A human-centered GIS approach to modeling mobility on southern Baffin Island, Nunavut, Canada. Journal of Field Archaeology, 41(6), 684-698. https://doi.org/10.1080/00934690.2016.1234897.

Bellavia's cost function:

$1/(N * ((atan(abs(x[adj])) * 180/pi) + 1))$

proposed by G. Bellavia, it measures abstract cost. Slope in degrees; N is a terrain factor (see above). **See**: Herzog I. (2020). Spatial Analysis Based on Cost Functions. In Gillings M, Haciguzeller P,

Lock G (eds), "Archaeological Spatial Analysis. A Methodological Guide.", Routledge: New York, 333-358 (with previous references).

Eastman's cost function:

$$1/((0.031 * (atan(abs(x[adj])) * 180/pi)^2 - 0.025 * (atan(abs(x[adj])) * 180/pi) + 1) * N)$$

proposed by J.R. Eastman, it measures abstract cost; slope in degrees. **See**: Vaissie E., Mobility of Paleolithic Populations: Biomechanical Considerations and Spatiotemporal Modelling, in PaleoAnthropology 2021 (1): 120-144 (with previous reference to Eastman 1999).

Pandolf et al.'s metabolic energy expenditure cost function (in Watts):

$$1/((1.5*W+2.0*(W+L)*(L/W)^2+N*(W+L)*(1.5*(V^2)+0.35*V*(abs(x[adj])*100)))*N)$$

where W is the walker's body weight (Kg), L is the carried load (in Kg), V is the velocity in m/s, N is a coefficient representing ease of movement on the terrain (see above). **Note** that if V is set to 0 by the user, it is internally worked out on the basis of the Tobler function for on-path hiking; therefore, V will not be considered constant throughout the analysed area, but will vary as function of the slope.

For this cost function, **see** Pandolf, K. B., Givoni, B., & Goldman, R. F. (1977). Predicting energy expenditure with loads while standing or walking very slowly. Journal of Applied Physiology, 43(4), 577-581. https://doi.org/10.1152/jappl.1977.43.4.577.

For the use of this cost function in a case study, **see** Rademaker, K., Reid, D. A., & Bromley, G. R. M. (2012). Connecting the Dots: Least Cost Analysis, Paleogeography, and the Search for Paleoindian Sites in Southern Highland Peru. In White D.A. & Surface-Evans S.L. (Eds.), Least Cost Analysis of Social Landscapes. Archaeological Case Studies (pp. 32-45). University of Utah Press; **see also** Herzog, I. (2013). Least-cost Paths - Some Methodological Issues, Internet Archaeology 36 (http://intarch.ac.uk/journal/issue36/index.html) with references. For the idea of using an hiking function inside an energetic cost function, **see** for instance White D.A., Prehistoric Trail Networks of the Western Papaguarie. A Multifaceted Least Cost Graph Theory Analysis. In White D.A. & Surface-Evans S.L. (Eds.), Least Cost Analysis of Social Landscapes. Archaeological Case Studies (pp. 188-206). University of Utah Press.

Note: in the returned charts, the cost is transposed from Watts to Megawatts (see, e.g., Rademaker et al 2012 cited above).

Pandolf et al.'s metabolic energy expenditure cost function with correction factor for downhill movements (in Watts):

$$ifelse(abs(x[adj]) * 100 > 0, 1/(1.5 * W + 2.0 * (W + L) * (L/W)^2 + N * (W + L) * (1.5 * V^2 + 0.35 * V * (abs(x[adj]) * 100))), 1/((1.5 * W + 2.0 * (W + L) * (L/W)^2 + N * (W + L) * (1.5 * V^2 + 0.35 * V * (abs(x[adj]) * 100))) - (N * ((abs(x[adj]) * 100) * (W + L) * V/3.5) -$$

$((W + L) * ((abs(x[adj]) * 100) + 6)^2/W) + (25 - V^2))))$

for the parameters W, L, V, and N, see above. If V is set to 0 by the user, it is internally worked out on the basis of the Tobler function for on-path hiking; therefore, V will not be considered constant throughout the analysed area, but will vary as function of the slope. For the correction factor applied to the Pandolf et al.'s cost function, **see** Yokota M., Berglund L.G., Santee W.R., Buller M.J., Hoyt R.W. (2004), Predicting Individual Physiological Responsed During Marksmanship Field Training Using an Updates Scenario-J Model. U.S. Army Research Institute of Environmental Medicine Technical Report T04-09. For an archaeological application of the Pandol et al's cost function with correction factor, **see** White D.A., Prehistoric Trail Networks of the Western Papaguarie. A Multi-faceted Least Cost Graph Theory Analysis. In White D.A. & Surface-Evans S.L. (Eds.), Least Cost Analysis of Social Landscapes. Archaeological Case Studies (pp. 188-206). University of Utah Press.

Note: in the returned charts, the cost is transposed from Watts to Megawatts (see, e.g., Rademaker et al 2012 cited above).

Minetti et al.'s metabolic energy cost function (in J/(kg*m)):

$1/(((280.5 * abs(x[adj])^5) - (58.7 * abs(x[adj])^4) - (76.8 * abs(x[adj])^3) + (51.9 * abs(x[adj])^2) + (19.6 * abs(x[adj])) + 2.5) * N)$

see Minetti A.E., Moia C., Roi G.S., Susta D., Ferretti G. (2002), Enery cost of walking and running at extreme uphilland downhill slopes, in Journal of Applied Physiology 93, 1039-1046. **Note** that this equation is valid for slopes in the range -0.5/0.5; outside this range, the function's output becomes counterintuitive, as noted in Paez et al. in Journal of Transport Geography 82 (2020) and Herzog I. (2013), Theory and practice of cost functions, in Contreras F., Farjas M., Melero F.J. (eds), "Fusion of cultures. Proceedings of the 38th annual conference on computer applications and quantitative methods in archaeology". BAR IS, 2494, 375-382. Oxford: Archaeopress. In the latter work, Herzog proposes to replace Minetti et al.'s equation with its 6th degrees polynomial approximation (see the Herzog's metabolic cost function below).

Herzog's metabolic cost function in J/(kg*m):

$1/(((1337.8 * abs(x[adj])^6) + (278.19 * abs(x[adj])^5) - (517.39 * abs(x[adj])^4) - (78.199 * abs(x[adj])^3) + (93.419 * abs(x[adj])^2) + (19.825 * abs(x[adj])) + 1.64) * N)$

see Herzog, I. (2016). Potential and Limits of Optimal Path Analysis. In A. Bevan & M. Lake (Eds.), Computational Approaches to Archaeological Spaces (pp. 179-211). New York: Routledge. Herzog suggests to use this as a 6th degree polynomial approximation of the Minetti et al's cost function (see above).

Van Leusen's metabolic energy expenditure cost function (in Watts):

$1/((1.5 * W + 2.0 * (W + L) * (L/W)^2 + N * (W + L) * (1.5 * (V^2) + 0.35 * V * ((abs(x[adj]) *$

$100) + 10))) * N)$

which modifies the Pandolf et al.'s equation; **see** Van Leusen, P. M. (2002). Pattern to process: methodological investigations into the formation and interpretation of spatial patterns in archaeological landscapes. University of Groningen. **Note** that, as per Herzog, I. (2013). Least-cost Paths - Some Methodological Issues, Internet Archaeology 36 (http://intarch.ac.uk/journal/issue36/index.html) and unlike Van Leusen (2002), in the above equation slope is expressed in percent and speed in m/s; also, in the last bit of the equantion, 10 replaces the value of 6 used by Van Leusen (as per Herzog 2013).
As explained above, if V is set to 0 by the user, it is internally worked out on the basis of the Tobler function for on-path hiking; therefore, V will not be considered constant throughout the analysed area, but will vary as function of the slope.

Note: in the returned charts, the cost is transposed from Watts to Megawatts.

Llobera-Sluckin's metabolic energy expenditure cost function (in KJ/m):

$1/((2.635 + (17.37 * abs(x[adj])) + (42.37 * abs(x[adj])^2) - (21.43 * abs(x[adj])^3) + (14.93 * abs(x[adj])^4)) * N)$

for which **see** Llobera M.-Sluckin T.J. (2007). Zigzagging: Theoretical insights on climbing strategies, Journal of Theoretical Biology 249, 206-217.

Ardigo et al.'s metabolic energy expenditure cost function (in J/(kg*m)):

$1/((1.866 * exp(4.911 * abs(x[adj])) * V^2 - 3.773 * exp(3.416 * abs(x[adj])) * V + (45.71 * abs(x[adj]^2) + 18.90 * abs(x[adj])) + 4.456) * N)$

see Ardigo L.P., Saibene F., Minetti A.E. (2003), The optimal locomotion on gradients: walking, running or cycling?, in Eur J Appl Physiol 90, 365-371. If V is set to 0 by the user, it is internally worked out on the basis of the Tobler function for on-path hiking; therefore, V will not be considered constant throughout the analysed area, but will vary as function of the slope.

Hare's metabolic energy expenditure cost function (in cal/km):

$1/((48 + 30/(1/(6 * exp(-3.5 * abs(x[adj] + 0.05))))) * N)$

see Hare T.S., Using Measures of Cost Distance in the Estimation of Polity Bounrdaries in the Post Classic Yautepec valley, Mexico, in Journal of Archaeological Science 31 (2004). Energetic expenditure is expressed in calories per km; walking speed is internally worked out from the DTM using the on-path Tobler's hiking function, which is expressed as its reciprocal; walking speed in km/h as per original Tobler's equation and as requested by Hare's function.

Note that the walking-speed-related cost functions listed above are used as they are, while the other functions are reciprocated. This is done since "gdistance works with conductivity rather than the

more usual approach using costs"; therefore "we need inverse cost functions" (Nakoinz-Knitter (2016). "Modelling Human Behaviour in Landscapes". New York: Springer, p. 183). As a consequence, if we want to estimate time, we have to use the walking-speed functions as they are since the final accumulated values will correspond to the reciprocal of speed, i.e. pace. In the other cases, we have to use 1/cost-function to eventually get cost-function/1.

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class)
- cost.surface: raster representing the cost-surface ('RasterLayer' class)
- accumulated.cost.raster: raster representing the accumualted cost ('RasterLayer' class)
- isolines: contour lines derived from the accumulated cost surface ('SpatialLinesDataFrame' class)
- LCPs: estimated least-cost paths ('SpatialLinesDataFrame' class)
- LCPs.back: estimated least-cost paths back to the origin ('SpatialLinesDataFrame' class)
- LCPs$length: length of each least-cost path (units depend on the unit used in the input DTM)
- LCPs.back$length: length of each least-cost path back to the origin (units depend on the unit used in the input DTM)
- dest.loc.w.cost: copy of the input destination location(s) dataset with a new variable ('cost') added; if the cost is expressed in terms of time, the 'cost' variable will store the time values in decimal numbers, while another variable named 'cost_hms' will store the time values in sexagesimmal numbers (hours, minutes, seconds)
- conductance: conductance 'Transitional Layer', returned because internally used by the `movenetw()` function

See Also

`get_elev_raster`, `movecorr`, `movebound`, `movealloc`, `movecomp`, `movenetw`, `moverank`

Examples

```
# load a sample Digital Terrain Model
data(volc)

# load a sample start location on the above DTM
data(volc.loc)

# load the sample destination locations on the above DTM
data(destin.loc)

# calculate walking-time isochrones based on the on-path Tobler's hiking function (default),
# setting the time unit to hours and the isochrones interval to 0.05 hour;
# also, since destination locations are provided,
# least-cost paths from the origin to the destination locations will be calculated
# and plotted; 8-directions move is used
```

```
result <- movecost(dtm=volc, origin=volc.loc, destin=destin.loc, move=8, breaks=0.05)


# same as above, but using the Irmischer-Clarke's hiking function (male, on-path)

result <- movecost(dtm=volc, origin=volc.loc, destin=destin.loc, funct="icmonp",
move=8, breaks=0.05)


# same as above, but using the 'cognitive slope'

result <- movecost(dtm=volc, origin=volc.loc, destin=destin.loc, funct="icmonp",
move=8, breaks=0.05, cogn.slp=TRUE)


# calculate accumulated cost surface and the least-cost path between the
# origin and one destination, and also calculate the LCP back to the origin

results <- movecost(dtm=volc, origin=volc.loc, destin=destin.loc[2,], move=8, return.base = TRUE)
```

`movenetw` *R function for calculating least-cost path network*

Description

The function provides the facility to calculate LCPs between multiple origins. Two types of networks are produced: one where each origin location is connected to all the others locations; one where only pairs of neighboring locations are connected. In other words, in the latter case, each location is connected to the location that is the nearest in terms of walking cost, either in terms of time or energy (or abstract cost), according to the selected cost function. Optionally, a raster representing the density of the first type of network can be produced.
Visit this LINK to access the package's vignette.

Usage

```
movenetw(
  dtm = NULL,
  origin,
  netw.type = "allpairs",
  studyplot = NULL,
  barrier = NULL,
  plot.barrier = FALSE,
  irregular.dtm = FALSE,
  funct = "t",
  move = 16,
```

```
  field = 0,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  lcp.dens = FALSE,
  transp = 0.5,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see movecost).
origin	locations from which the network of least-cost paths is calculated (SpatialPointsDataFrame class).
netw.type	type of network to be calculated: 'allpairs' (default) or 'neigh' (see 'Details').
studyplot	polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see movecost); NULL is default.
barrier	area where the movement is inhibited (SpatialLineDataFrame or SpatialPolygonDataFrame class) (see movecost).
plot.barrier	TRUE or FALSE (default) if the user wants or does not want the barrier to be plotted (see movecost).
irregular.dtm	TRUE or FALSE (default) if the input DTM features irregular margins (see movecost).
funct	cost function to be used (for details on each of the following, see movecost): **-functions expressing cost as walking time-** **t** (default) uses the on-path Tobler's hiking function; **tofp** uses the off-path Tobler's hiking function; **mp** uses the Marquez-Perez et al.'s modified Tobler's function; **icmonp** uses the Irmischer-Clarke's hiking function (male, on-path); **icmoffp** uses the Irmischer-Clarke's hiking function (male, off-path); **icfonp** uses the Irmischer-Clarke's hiking function (female, on-path); **icfoffp** uses the Irmischer-Clarke's hiking function (female, off-path); **ug** uses the Uriarte Gonzalez's walking-time cost function; **ma** uses the Marin Arroyo's walking-time cost function; **alb** uses the Alberti's Tobler hiking function modified for pastoral foraging excursions; **gkrs** uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function; **r** uses the Rees' hiking function; **ks** uses the Kondo-Seino's hiking function; **trp** uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;
a uses the Ardigo et al.'s metabolic energy expenditure cost function;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see `movecost`).

`move` number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).

`field` value assigned to the cells coinciding with the barrier (0 by default) (see `movecost`.

`cogn.slp` TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see `movecost`).

`sl.crit` critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see `movecost`).

`W` walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see `movecost`).

`L` carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see `movecost`).

`N` coefficient representing ease of movement (1 by default) (see `movecost`).

`V` speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function (see `movecost`).

`z` zoom level for the elevation data downloaded from online sources (from 0 to 15; 9 by default) (see `movecost` and `get_elev_raster`).

`lcp.dens` TRUE or FALSE (default) if the user wants or does not want the least-cost paths density raster to be produced.

`transp` set the transparency of the slopeshade raster that is plotted over the DTM (0.5 by default).

`export` TRUE or FALSE (default) if the user wants or does not want the LCPs network to be exported as a shapefile, and the LCPs network density as a GeoTiff; the DTM is exported only if it was not provided by the user and downloaded by the function from online sources.

Details

Like `movecost()`, the function just requires an input DTM ('RasterLayer' class) and an origin dataset ('SpatialPointsDataFrame' class). If a DTM is not provided, `movenetw()` downloads elevation data from online sources for the area enclosed by the polygon fed via the `studyplot` parameter (see `movecost` for more details). Under the hood, `movenetw()` relies on `movecost()` and implements the same cost functions: see the help documentation of `movecost()` for further information.

`movenetw()` produces a plot representing the input DTM overlaid by a slopeshade raster, whose transparency can be adjusted using the `transp` parameter. On the rendered plot, the LPCs network ('SpatialLinesDataFrame' class) is represented by black lines. To calculate the network connecting all the locations, the user may want to set the `netw.type` parameter to `allpairs` (which is the default value). If the user wants to calculate the network connecting neighbouring locations, the `netw.type` parameter is to be set to `neigh`. Optionally, by setting the `lcp.dens` parameter to `TRUE`, the function produces a raster representing the density of the LCPs connecting each location to all the other locations. The raster, which is rendered overlaid to a slopeshade visualization, expresses the density of LCPs as percentages. The percentages are calculated in relation to the maximum number of LCPs passing through the same cell stored in the raster. A density raster expressing counts is NOT rendered BUT is returned by the function. The density raster retains the cell size and coordinate system of the input DTM.

The function returns a list storing the DTM (only in case this has not been fed into the function but acquired online), a list of LCPs split by origin, a SpatialLineDataFrame representing the merged LCPs, two rasters representing the LCPs network density expressed as counts and percentages respectively, and cost matrices. As for the latter, if the selected cost function defines cost as walking time, two matrices are returned, one expressing time in minutes, one in hours (**note** that the values are in decimal format). If the selected cost function expresses cost differently (i.e., energy or abstract cost), the two above mentioned cost matrices will be set to NULL, and a third cost matrix will store all the pair-wise costs.

The above mentioned data (DTM, LCPs, network density) can be exported by setting the `export` parameter to `TRUE`. The LCPs network (exported as a shapefile) and the density raster (as a GeoTiff) will bear a suffix indicating the used cost function.

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class); returned only if acquired online
- LCPs.netw: list containing the LCPs ('SpatialLinesDataFrame' class) split by origin
- LCPs.netw.merged: 'SpatialLinesDataFrame' corresponding to the merged LCPs
- LCPs.netw.neigh: list containing the LCPs between neighboring locations ('SpatialLinesDataFrame' class) split by origin
- LCPs.netw.merged: 'SpatialLinesDataFrame' corresponding to the merged LCPs between neighboring locations
- LCPs.density.count: raster ('RasterLayer' class) representing the counts of LCPs on each raster's cell

- LCPs.density.perc: same as the preceding, but re-expressing the counts as percentages
- cost.matrix.min: matrix of cost between locations, expressing cost in minutes
- cost.matrix.hr: matrix of cost between locations, expressing cost in hours
- cost.matrix: matrix of cost between locations, expressing cost either in energy or abstract cost, depending on the used cost function

See Also

movecost

Examples

```
# load a sample Digital Terrain Model
data(volc)


# load the sample destination locations on the above DTM
data(destin.loc)


# calculate the least-cost path network between neighboring locations
# using the Tobler's hiking function (for on-path walking)

result <- movenetw(dtm=volc, origin=destin.loc, move=8, funct="t", netw.type="neigh")
```

`moverank` *R function for calculating sub-optimal least-cost paths bewteen an origin and a destination location*

Description

The function provides the facility to calculate the LCP between an origin and a destination location and (more importantly) to work out the first five sub-optimal LCPs between those locations. The underlying idea is the following: given two locations, we can calculate the least-costly path between them; but, if we disregard that LCP, what path would be the second least costly? And if we in turn disregard those first two, what the third least costly path would be? The same reasoning holds for all the subsequent n-th LCPs. Under the hood, `moverank()` rests on `movecost` and implements the same cost functions. See the help documentation of `movecost()` for further details.
Visit this LINK to access the package's vignette.

Usage

```
moverank(
  dtm = NULL,
  origin,
  destin,
  studyplot = NULL,
  barrier = NULL,
  plot.barrier = FALSE,
  irregular.dtm = FALSE,
  funct = "t",
  time = "h",
  lcp.n = 3,
  move = 16,
  cogn.slp = FALSE,
  sl.crit = 10,
  W = 70,
  L = 0,
  N = 1,
  V = 1.2,
  z = 9,
  use.corr = FALSE,
  leg.pos = "topright",
  leg.cex = 0.55,
  add.chart = FALSE,
  bubble.cex = 0.5,
  transp = 0.5,
  export = FALSE
)
```

Arguments

dtm	Digital Terrain Model (RasterLayer class); if not provided, elevation data will be acquired online for the area enclosed by the 'studyplot' parameter (see movecost).
origin	location from which least-cost path(s) is calculated (SpatialPointsDataFrame class).
destin	location(s) to which least-cost path(s) is calculated (SpatialPointsDataFrame class).
studyplot	polygon (SpatialPolygonDataFrame class) representing the study area for which online elevation data are acquired (see movecost); NULL is default.
barrier	area where the movement is inhibited (SpatialLineDataFrame or SpatialPolygonDataFrame class) (see movecost).
plot.barrier	TRUE or FALSE (default) if the user wants or does not want the barrier to be plotted (see movecost).
irregular.dtm	TRUE or FALSE (default) if the input DTM features irregular margins (see movecost).

`funct` cost function to be used (for details on each of the following, see `movecost`):

-functions expressing cost as walking time-
t (default) uses the on-path Tobler's hiking function;
tofp uses the off-path Tobler's hiking function;
mp uses the Marquez-Perez et al.'s modified Tobler's function;
icmonp uses the Irmischer-Clarke's hiking function (male, on-path);
icmoffp uses the Irmischer-Clarke's hiking function (male, off-path);
icfonp uses the Irmischer-Clarke's hiking function (female, on-path);
icfoffp uses the Irmischer-Clarke's hiking function (female, off-path);
ug uses the Uriarte Gonzalez's walking-time cost function;
ma uses the Marin Arroyo's walking-time cost function;
alb uses the Alberti's Tobler hiking function modified for pastoral foraging excursions;
gkrs uses the Garmy, Kaddouri, Rozenblat, and Schneider's hiking function;
r uses the Rees' hiking function;
ks uses the Kondo-Seino's hiking function;
trp uses the Tripcevich's hiking function;

-functions for wheeled-vehicles-
wcs uses the wheeled-vehicle critical slope cost function;

-functions expressing abstract cost-
ree uses the relative energetic expenditure cost function;
b uses the Bellavia's cost function;
e uses the Eastman's cost function;

-functions expressing cost as metabolic energy expenditure-
p uses the Pandolf et al.'s metabolic energy expenditure cost function;
pcf uses the Pandolf et al.'s cost function with correction factor for downhill movements;
m uses the Minetti et al.'s metabolic energy expenditure cost function;
hrz uses the Herzog's metabolic energy expenditure cost function;
vl uses the Van Leusen's metabolic energy expenditure cost function;
ls uses the Llobera-Sluckin's metabolic energy expenditure cost function;
a uses the Ardigo et al.'s metabolic energy expenditure cost function;
h uses the Hare's metabolic energy expenditure cost function (for all the mentioned cost functions, see `movecost`).

`time` time-unit expressed by the accumulated raster if Tobler's and other time-related cost functions are used; h' for hour, 'm' for minutes.

`lcp.n` number of LCPs rendered in the output plot (min=1, max=6; 3 by default; the 1st LCP is the optimal one, while the LCPs from the 2nd to the 6th are the sub-optimal ones).

`move` number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), 16 (knight and one-cell queen moves; default).

cogn.slp	TRUE or FALSE (default) if the user wants or does not want the 'cognitive slope' to be used in place of the real slope (see movecost).
sl.crit	critical slope (in percent), typically in the range 8-16 (10 by default) (used by the wheeled-vehicle cost function; see movecost).
W	walker's body weight (in Kg; 70 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
L	carried load weight (in Kg; 0 by default; used by the Pandolf's and Van Leusen's cost function; see movecost).
N	coefficient representing ease of movement (1 by default) (see movecost).
V	speed in m/s (1.2 by default) (used by the Pandolf et al.'s, Pandolf et al.s with correction factor, Van Leusen's, and Ardigo et al.'s cost function; if set to 0, it is internally worked out on the basis of Tobler on-path hiking function (see movecost).
z	zoom level for the elevation data downloaded from online sources (from 0 to 15; 9 by default) (see movecost and get_elev_raster).
use.corr	TRUE or FALSE (default) is the user wants or does not want the least-cost corridor raster to be rendered in place of the input DTM.
leg.pos	set the position of the legend in rendered plot; 'topright' by default (other options: "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center").
leg.cex	set the size of the labels used in the legend displayed in the rendered plot (0.55 by default).
add.chart	TRUE or FALSE (default) is the user wants or does not want a bubble chart visualising LCPs length vs rank vs cost to be rendered.
bubble.cex	set the size of the labels reporting the LCPs cost in the bubble chart (0.5 by default).
transp	set the transparency of the slopeshade raster that is plotted over the least-cost corridor raster (0.5 by default).
export	TRUE or FALSE (default) if the user wants or does not want the output to be exported; if TRUE, the least-cost corridor and the DTM (if not provided by the user but acquired online) are expoerted as a GeoTiff file, while the LCPs as a shapefile layer. All the exported files (excluding the DTM) will bear a suffix corresponding to the cost function selected by the user.

Details

Internally, moverank() uses movecost() to generate the first (optimal) LCP. In a second iteration, the optimal LCP is internally used as barrier (see movecost) when calculating the 2nd LCPs. Then, in a third iteration, the two previously generated LCPs are used as barriers when working out the 3rd LCPs. The process repeats along the same lines until the 6th LCP is calculated. The 1st LCP is deemed to represent the optimal path (cost-wise) between the two locations, while the 2nd-to-5th LCPs are deemed to represent progressively sub-optimal paths.

It is worth noting that it may happen that some LCP will cross another one; this cannot be anticipated and is context dependent. In those cases, the user may want to set the move parameter to 8

(see the section about inhibition of movement in the help documentation of movecost).

The function provides the facility to render the LCPs either on the input DTM or on the least-cost corridor between the two locations. The second option can be obtained by setting the use.corr parameter to TRUE. Also, by setting the add.chart parameter to TRUE, the function renders a bubble chart that plots the LCPs length against their rank, while the size of the bubbles is proportional to the cost. All the LCPs will be plotted.

By setting the export parameter to TRUE, the LCPs, the DTM (if acquired online), and the least-cost corridor (if obtained by setting the use.corr parameter to TRUE), will be exported: the DTM and least-cost corridor as a raster layer, the LCPs as shapefile layer. The LCPs and the least-cost corridor files will be given a suffix indicating which cost function has been used.

Value

The function returns a list storing the following components

- dtm: Digital Terrain Model ('RasterLayer' class); returned only if not provided by the user and acquired online instead
- LCPs: least-cost paths ('SpatialLinesDataFrame' class) ranked from 1 (optimal) to 6 (sub-optimal LCPs)
- lc.corr: least-cost corridor between the origin and destination location ('RasterLayer' class); returned if the use.corr parameter is set to TRUE

See Also

movecost

Examples

```
# load a sample Digital Terrain Model
data(volc)


# load the sample destination locations on the above DTM
data(destin.loc)

# calculate the optimal and sub-optimals LCPs between two locations
#result <- moverank(volc, destin.loc[1,], destin.loc[4,], move=8, funct="t")
```

springs *Dataset: location of springs in Malta*

Description

A SpatialPointsDataFrame representing the location of spring-related toponyms in Malta.

Usage

```
data(springs)
```

Format

SpatialPointsDataFrame

volc *Dataset: raster dataset representing the elevation of the volcano Maunga Whau (Auckland, New Zealand)*

Description

A RasterLayer representing a DTM of the volcano Maunga Whau (Auckland, New Zealand).

Usage

```
data(volc)
```

Format

RasterLayer

volc.loc *Dataset: location on the volcano Maunga Whau (Auckland, New Zealand)*

Description

A SpatialPointsDataFrame representing a spot on the volcano Maunga Whau (Auckland, New Zealand).

Usage

```
data(volc.loc)
```

Format

SpatialPointsDataFrame

Index

∗ **datasets**
destin.loc, 2
Etna_boundary, 3
Etna_end_location, 3
Etna_start_location, 4
malta_dtm_40, 4
springs, 48
volc, 48
volc.loc, 48
∗ **movealloc**
movealloc, 5
∗ **movebound**
movebound, 9
∗ **movecomp**
movecomp, 13
∗ **movecorr**
movecorr, 18
∗ **movecost**
movecost, 23
∗ **movenetw**
movenetw, 39
∗ **moverank**
moverank, 43

destin.loc, 2

Etna_boundary, 3
Etna_end_location, 3
Etna_start_location, 4

get_elev_raster, *7*, *11*, *16*, *20*, *26*, *29*, *38*, *41*, *46*

malta_dtm_40, 4
movealloc, 5, *38*
movebound, 9, *38*
movecomp, 13, *38*
movecorr, 18, *38*
movecost, *5–20*, *22, 23*, 23, *40–47*
movenetw, *38*, 39
moverank, *38*, 43

par, *26*

springs, 48

volc, 48
volc.loc, 48