Package: movecost (via r-universe)

October 31, 2026

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 3.1.1

Description Provides the facility to calculate non-isotropic accumulated cost surfaces, least-cost paths, least-cost corridors, least-cost networks, ranked alternative paths, cost allocation and cost boundaries, using a number of human-movement-related cost functions that can be selected by the user. The terrain factor can be set as a single value or vary across the landscape via a raster of terrain factors or terrain classes. The package is built around a compute-once design: a single cost surface object is created first and then reused by every analysis function, avoiding redundant computation. Visualisation is fully decoupled from computation and is provided through 'ggplot2' methods that can be invoked, customised, and re-invoked at any time without re-running any analysis. 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 (>= 4.1.0)

Imports terra (>= 1.7-0), sf (>= 1.0-9), igraph (>= 1.4.0), ggplot2 (>= 3.4.0), utils, grDevices

Suggests elevatr (>= 0.99.0), testthat (>= 3.0.0), knitr, rmarkdown

License GPL (>= 2)

Encoding UTF-8

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Gianmarco Alberti [aut, cre]

Maintainer Gianmarco Alberti <gianmarcoalberti@gmail.com>

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

Date/Publication 2026-10-31 12:49:44 UTC

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

RemoteRef HEAD

RemoteSha 700f45cf8d9b3f6a6fd2afa5946fd896f0820f20

Contents

mc_accum . . . 2
mc_alloc . . . 4
mc_boundary . . . 5
mc_comp . . . 6
mc_corridor . . . 8
mc_cost_functions . . . 11
mc_dtm . . . 18
mc_export . . . 19
mc_network . . . 20
mc_paths . . . 22
mc_rank . . . 23
mc_save . . . 25
mc_surface . . . 25
mc_terrain_factors . . . 30
movecost-data . . . 31
plot.movecost_accum . . . 33
plot.movecost_alloc . . . 34
plot.movecost_boundary . . . 35
plot.movecost_comp . . . 35
plot.movecost_corridor . . . 36
plot.movecost_network . . . 37
plot.movecost_paths . . . 38
plot.movecost_rank . . . 39
plot.movecost_surface . . . 39

Index **41**

`mc_accum` *Accumulated cost surface and isolines around one or more origins*

Description

Computes the anisotropic accumulated cost of movement around the input origin location(s), reusing the graph stored in a `mc_surface` object (nothing is recomputed). If more than one origin is supplied, the returned surface stores, for each cell, the minimum accumulated cost across origins (as in movecost <= 2.x).

Usage

```
mc_accum(surface, origin, time = "h", breaks = NULL)
```

Arguments

surface	a movecost_surface object created by mc_surface.
origin	origin location(s) (sf points, SpatVector, or legacy SpatialPointsDataFrame).
time	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).
breaks	isoline interval; if NULL (default), one tenth of the range of accumulated values.

Details

Isolines (contour lines) are derived from the accumulated surface at an interval set by breaks; if no value is supplied, the interval defaults to one tenth of the range of the accumulated values.

For cost functions expressing cost as walking time, the accumulated values are returned in the unit selected via time ("h" hours, default, or "m" minutes); for the energy functions, values are expressed as accumulated energy (kJ for the Pandolf, Pandolf-corrected, Van Leusen, and Llobera-Sluckin functions; J/kg for the Minetti, Herzog, and Ardigo functions; kcal for Hare's function); abstract-cost functions keep their native unit (see the cost_unit column of mc_cost_functions).

Value

An object of class movecost_accum, a list with components

- accum: accumulated cost surface (SpatRaster), masked to the DTM (NoData cells, e.g. sea, are NA);
- isolines: contour lines (sf lines) with a level attribute;
- origin: the origin location(s) (sf);
- breaks, levels, time, funct, funct.label, cost.unit: metadata used by the plot method.

Plot it at any time, and as many times as desired, with plot() (see plot.movecost_accum); no computation is re-run.

See Also

mc_surface, plot.movecost_accum, mc_export

Examples

```
dtm <- mc_volc()
start <- mc_volc_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)
acc <- mc_accum(surf, origin = start, breaks = 0.05)


plot(acc)                # raster + isolines
```

```
plot(acc, type = "contour")  # isolines only
```

`mc_alloc` *Cost allocation (Thiessen-like polygons based on movement cost)*

Description

Partitions the landscape into zones of pertinence, assigning each DTM cell to the origin location from which it can be reached at the smallest accumulated cost. The result is the cost-based analogue of Voronoi/Thiessen polygons.

Usage

```
mc_alloc(surface, origin, time = "h", breaks = NULL)
```

Arguments

`surface`	a `movecost_surface` object created by `mc_surface`.
`origin`	origin location(s) (sf points, SpatVector, or legacy SpatialPointsDataFrame); at least two.
`time`	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).
`breaks`	optional isoline interval for the minimum-cost surface; if NULL (default), one tenth of its range. Isolines are returned for optional display in the plot method.

Details

All the accumulated cost surfaces are obtained with a single multi-source graph query on the `mc_surface` object: with n origins, movecost <= 2.x performed n full conductance-matrix constructions plus n accumulation runs, whereas this implementation performs none of the former and one batched run of the latter, making allocation over many sites orders of magnitude faster.

Value

An object of class `movecost_alloc`, a list with components

- `alloc`: allocation zones (`SpatRaster`; cell value = index of the prevailing origin);
- `zones`: the same zones as polygons (`sf`) with an `origin_id` attribute;
- `accum.min`: minimum accumulated cost across origins (`SpatRaster`);
- `isolines`: isolines of `accum.min` (`sf`);
- `origin`: the origin locations (`sf`);
- metadata used by the plot method.

Plot it at any time with `plot()` (see `plot.movecost_alloc`).

See Also

mc_surface, plot.movecost_alloc

Examples

```
dtm <- mc_volc()
destin <- mc_destin_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)
al <- mc_alloc(surf, origin = destin)

plot(al)                    # allocation zones
plot(al, isolines = TRUE)   # with cost isolines overlaid
```

---

mc_boundary *Cost-limit boundaries around origin locations*

Description

Delineates, around each origin location, the area reachable within a given cost limit (for instance, the 30-minute walking catchment), reusing the graph stored in a mc_surface object. The accumulated cost surfaces for all origins are obtained with a single multi-source graph query, so the function scales efficiently with the number of origins (movecost <= 2.x recomputed the whole conductance matrix for each call).

Usage

```
mc_boundary(surface, origin, limit = NULL, time = "h")
```

Arguments

surface	a movecost_surface object created by mc_surface.
origin	origin location(s) (sf points, SpatVector, or legacy SpatialPointsDataFrame).
limit	cost limit defining the boundary, in the unit of the cost function (for time-based functions, in the unit selected via time); if NULL (default), one tenth of the range of accumulated values across all origins is used.
time	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).

Details

Unlike movecost <= 2.x, which derived the boundary from contour lines of a combined minimum-cost surface, version 3.0 polygonises each origin's reachable area directly (cells with accumulated cost <= limit), which yields closed polygons by construction and allows area and perimeter to be computed exactly. Overlapping catchments of different origins are preserved as separate (overlapping) polygons.

Value

An object of class `movecost_boundary`, a list with components

- `boundaries`: `sf` polygons (one feature per origin) with `origin_id`, `limit`, `area` (m^2) and `perimeter` (m) attributes;
- `accum`: the minimum accumulated cost surface across origins (`SpatRaster`), useful for plotting;
- `origin`: the origin location(s) (`sf`);
- metadata used by the plot method.

Plot it at any time with `plot()` (see `plot.movecost_boundary`).

See Also

`mc_surface`, `plot.movecost_boundary`

Examples

```
dtm <- mc_volc()
destin <- mc_destin_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)
# 5-minute walking catchments around three locations
bnd <- mc_boundary(surf, origin = destin[1:3, ], limit = 5, time = "m")

bnd$boundaries     # polygons with area and perimeter
plot(bnd)
```

---

mc_comp *Compare least-cost paths produced by different cost functions*

Description

Computes, for the same origin and destination(s), the least-cost paths produced by two or more cost functions, so that route choice and cost can be compared across cost models. One cost surface is built per function (this is unavoidable, since each function defines its own graph weights), but each is built once and all the remaining computation reuses it.

Usage

```
mc_comp(
  dtm,
  origin,
  destin,
  functs,
```

```
  move = 16,
  return.base = FALSE,
  time = "h",
  ...
)
```

Arguments

dtm	Digital Terrain Model (SpatRaster, RasterLayer, or file path), as in mc_surface.
origin	single origin location (sf points, SpatVector, or legacy SpatialPointsDataFrame).
destin	destination location(s) (same accepted classes).
functs	character vector of cost-function codes to compare (see mc_cost_functions), e.g. c("t", "ks", "hrz").
move	number of directions in which cells are connected (4, 8, or 16; default 16).
return.base	TRUE or FALSE (default): also compute the paths back to the origin for each cost function.
time	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).
...	further parameters passed to mc_surface (N, W, L, V, sl.crit, barrier, field, cogn.slp, topo.dist, terrain, terrain.table). A terrain raster is aligned to the DTM once and then shared by all the compared functions; functions that do not take a terrain factor (the off-path functions and Alberti's) ignore it, with a warning.

Details

Note that costs produced by functions of different families (walking time, energy, abstract) are expressed in different units and are comparable in terms of route geometry, not of cost magnitude; the cost table returned reports the unit alongside each value.

Value

An object of class movecost_comp, a list with components

- paths: sf lines for all functions, with funct, destination, cost, cost_unit, and length attributes;
- paths.back: return paths (NULL unless return.base = TRUE);
- cost.table: data frame summarising cost and length per function and destination;
- origin, destin: the input locations (sf);
- metadata used by the plot method.

Plot it at any time with plot() (see plot.movecost_comp); plot(x, type = "chart") compares the path length and the cost across functions, as boxplots of their distribution across destinations when three or more destinations are used (the add.chart output of movecost <= 2.x), or as bars otherwise.

See Also

mc_surface, mc_paths, plot.movecost_comp

Examples

```
dtm <- mc_volc()
start <- mc_volc_loc()
destin <- mc_destin_loc()

cmp <- mc_comp(dtm, origin = start, destin = destin[2, ],
               functs = c("t", "ks", "mp"), move = 8)

cmp$cost.table
plot(cmp)                  # routes on the terrain
plot(cmp, type = "chart")  # comparative bar chart
```

`mc_corridor` *Least-cost corridor between locations*

Description

Computes the least-cost corridor between point locations, that is, the band of terrain within which movement between the locations is comparatively cheap. The corridor is built from accumulated cost surfaces produced by the reused `mc_surface` graph. Two formulations are offered through the `method` argument; they coincide for isotropic cost functions and differ only when the cost function is anisotropic (direction-dependent, for instance the Tobler hiking function).

Usage

```
mc_corridor(
  surface,
  a,
  b = NULL,
  method = c("reach", "through"),
  lcp = TRUE,
  rescale = FALSE,
  time = "h"
)
```

Arguments

`surface`	a `movecost_surface` object created by `mc_surface`.
`a`	first location, or (if `b` is NULL) a set of locations between whose unique pairs corridors are computed and summed (sf points, SpatVector, or legacy SpatialPointsDataFrame).
`b`	second location (same accepted classes); NULL by default.
`method`	`"reach"` (default) for the classic symmetric corridor (sum of two outward accumulated surfaces), or `"through"` for the directional A-to-B near-optimal-route band (see Details).

`lcp`	TRUE (default) or FALSE: when exactly two locations are supplied, also compute the least-cost paths A-to-B and B-to-A and overlay them on the plot. Ignored when more than two locations are supplied.
`rescale`	TRUE or FALSE (default): rescale the corridor values to the 0-1 range (useful for comparing corridors computed with different cost functions).
`time`	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).

Details

method = "reach" (default; the classic least-cost corridor):
for every cell x the value is $cost(A \to x) + cost(B \to x)$, the sum of the accumulated cost spreading outward from each location. This is the definition used in the GIS literature and in movecost <= 2.x: "the cost of a cell is the total cost to reach it from all the analysed locations" (Mitchell A. 2012, The ESRI Guide to GIS Analysis, Vol. 3, Esri Press, 257-259). The surface is symmetric in A and B, so it describes a direction-free corridor (a band of ground that is cheap to reach from both endpoints), which matches the usual conception of a corridor as bidirectional. Use this for the standard corridor analysis, for reproducibility against movecost <= 2.x and other GIS tools, and whenever the corridor is conceived as a two-way swath between places.

method = "through" (directional, near-optimal-route band):
for every cell x the value is $cost(A \to x) + cost(x \to B)$, the cost of a journey from A to B that is constrained to pass through x (the second term is computed on the reversed graph). The minimum of this surface equals exactly the cost of the least-cost path from A to B, and cells whose value is within a chosen tolerance of that minimum form the band of near-optimal A-to-B routes. Unlike "reach", this surface is direction-specific: the (A, B) corridor differs from the (B, A) corridor under anisotropic costs. Use this when the interest is explicitly in one-way journeys from A to B and in identifying routes close to the optimal one in cost terms.

Why the two can differ: with an anisotropic cost function the cost of moving from B to x is not the cost of moving from x to B. The "reach" surface adds two outward ("from") costs and is symmetric; the "through" surface adds an outward and an inward cost and is directional. For symmetric (isotropic) cost functions the two are identical. As a concrete illustration, on sloped terrain with A low and B high the minimum of the "reach" surface corresponds to the cheaper of the two one-way trips (typically the downhill direction), whereas the minimum of the "through" surface is exactly the A-to-B (uphill) least-cost-path cost.

Least-cost path overlay: when exactly two locations are supplied (`a` and `b`) and `lcp = TRUE` (default), the least-cost paths from A to B and from B to A are also computed and returned, and are drawn over the corridor by the plot method, as in movecost <= 2.x. The two paths differ under anisotropic costs.

More than two locations: if a set of locations is supplied via `a` (with `b` left NULL), corridors are computed between all unique pairs and summed, as in movecost <= 2.x; the individual pairwise corridors are also returned. The accumulated surfaces are obtained from one multi-source graph query (`"reach"`) or two (`"through"`), rather than two per pair. No least-cost paths are overlaid in this case.

Value

An object of class `movecost_corridor`, a list with components

- `corridor`: the corridor surface (`SpatRaster`);
- `corridors`: for more than two locations, a list of the individual pairwise corridor surfaces (NULL for the two-location case);
- `locations`: the input locations (`sf`);
- `lcp.AtoB`, `lcp.BtoA`: the two least-cost paths (`sf` lines with `cost` and `length`) for the two-location case with `lcp = TRUE`, else NULL;
- `lcp.cost`: for `method = "through"` on two locations, the A-to-B least-cost-path cost (equal to the corridor minimum); NULL otherwise;
- `method`: the formulation used;
- metadata used by the plot method.

Plot it at any time with `plot()` (see `plot.movecost_corridor`).

References

Mitchell, A. (2012). The ESRI Guide to GIS Analysis. Vol. 3: Modelling Suitability, Movement, and Interaction. New York: Esri Press, 257-259.

See Also

`mc_surface`, `plot.movecost_corridor`, `mc_paths`

Examples

```
dtm <- mc_volc()
destin <- mc_destin_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)

# classic (symmetric) least-cost corridor, with the two LCPs overlaid
corr <- mc_corridor(surf, a = destin[1, ], b = destin[4, ])
plot(corr)

# directional A-to-B near-optimal-route band
corr2 <- mc_corridor(surf, a = destin[1, ], b = destin[4, ], method = "through")
plot(corr2)
```

`mc_cost_functions` *The movecost cost-function registry*

Description

`mc_cost_functions()` returns a data frame describing every cost function implemented in the package. The registry is the single internal source from which all analysis functions draw their cost models: each function is defined once, here, and reused everywhere (in movecost <= 2.x the same definitions were repeated across functions; the registry eliminates that duplication).

Usage

```
mc_cost_functions()
```

Details

Each cost function maps the terrain slope between two adjacent cells (expressed as rise over run, signed: positive uphill, negative downhill, and computed in the direction of movement) to either a walking speed or the reciprocal of a per-metre cost. In what follows, **x** stands for slope as rise/run calculated between adjacent cells in the direction of travel.

Functions expressing cost as walking time (speed in km/h):

Tobler's hiking function (on-path) (`"t"`):
$(6 * exp(-3.5 * abs(x + 0.05))) * (1/N)$
Tobler W. (1993), Three Presentations on Geographical Analysis and Modeling, NCGIA Technical Report 93-1.

Tobler's hiking function (off-path) (`"tofp"`):
$(6 * exp(-3.5 * abs(x + 0.05))) * 0.6$
as per Tobler's indication, the off-path walking speed is reduced by 0.6. This function does not take the terrain factor N, since it already embeds off-path walking; to apply a terrain factor, use the on-path version (`"t"`).

Marquez-Perez et al.'s modified Tobler hiking function (`"mp"`):
$(4.8 * exp(-5.3 * abs((x * 0.7) + 0.03))) * (1/N)$
Marquez-Perez J., Vallejo-Villalta I., Alvarez-Francoso J.I. (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 hiking function (male, on-path) (`"icmonp"`):
$((0.11 + exp(-(x * 100 + 5)^2/(2 * 30^2))) * 3.6) * (1/N)$
Irmischer I.J., Clarke K.C. (2018), Measuring and modeling the speed of human navigation, Cartography and Geographic Information Science, 45(2), 177-186, doi:10.1080/15230406.2017.1292150.

The original functions express speed in m/s; they are multiplied by 3.6 to obtain km/h for consistency with the other Tobler-related cost functions. Slope is the signed percent slope (100 * elevation change / distance, negative downhill), as in equations 2-5 of the source, so that the on-path speed peaks at -5% and the off-path speed at -2% (see also Herzog 2020, Table 18.2). **Note**: movecost <= 3.1.0 applied the absolute value to the slope, which removed this asymmetry; fixed in 3.1.1, so results for downhill movement differ from earlier versions.

Irmischer-Clarke's hiking function (male, off-path) (`"icmoffp"`):
$(0.11 + 0.67 * exp(-(x * 100 + 2)^2/(2 * 30^2))) * 3.6$
the Gaussian denominator is $2 * 30^2$ (=1800), the 2*sigma^2 form of equation 4 of Irmischer & Clarke (2018), shared by the on-path (equation 2) and female (equation 5) variants; signed percent slope, as for the on-path version (fixed in 3.1.1). This function does not take the terrain factor N, since it already embeds off-path walking; to apply a terrain factor, use the on-path version (`"icmonp"`).

Irmischer-Clarke's hiking function (female, on-path) (`"icfonp"`):
$((0.95 * (0.11 + exp(-(x * 100 + 5)^2/(2 * 30^2)))) * 3.6) * (1/N)$
signed percent slope (fixed in 3.1.1; see the male on-path version).

Irmischer-Clarke's hiking function (female, off-path) (`"icfoffp"`):
$(0.95 * (0.11 + 0.67 * exp(-(x * 100 + 2)^2/(2 * 30^2)))) * 3.6$
signed percent slope (fixed in 3.1.1; see the male on-path version). This function does not take the terrain factor N, since it already embeds off-path walking; to apply a terrain factor, use the on-path version (`"icfonp"`).

Uriarte Gonzalez's walking-time cost function (`"ug"`):
$1/((0.0277 * (abs(x) * 100) + 0.6115) * N)$
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, doi:10.1201/9780203881613.ch12. The function originally expresses walking time in seconds; its reciprocal is used internally so that the accumulated value corresponds to time. Unlike in the original formulation, the pixel resolution is not embedded in the formula because the graph machinery accounts for the actual distance between cell centres. The slope (P, in percent) is taken as a magnitude, as in the source, where the function is a regression on data of Gilman and Thornes (1985) relating the number of contour lines crossed to the distance covered in 12 minutes, irrespective of the direction of travel, and is applied to a slope layer; the function is therefore symmetric.

Marin Arroyo's walking-time cost function (`"ma"`):
$ifelse(x < 0, 1/((0.6 * ((abs(x) * 100)/23 + 1)) * N), 1/((0.6 * ((abs(x) * 100)/11 + 1)) * N))$
Marin Arroyo A.B. (2009), The use of optimal foraging theory to estimate Late Glacial site catchment areas from a central place: the case of eastern Cantabria, Spain, Journal of Anthropological Archaeology 28, 27-36. **Note**: in movecost <= 2.x the downhill branch could never be selected because the test was applied to the absolute slope; version 3.0 evaluates the signed slope, so downhill movement now correctly uses the gentler divisor (23), as in the original publication. Results for downhill movement therefore differ (correctly) from movecost <= 2.x.

Alberti's Tobler hiking function modified for pastoral foraging excursions (`"alb"`):
$(6 * exp(-3.5 * abs(x + 0.05))) * 0.25$
Alberti G., Grima R., Vella N.C. (2020), Locating potential pastoral foraging routes in Malta through the use of a Geographic Information System, in French C., Hunt C.O., Grima R., McLaughlin R., Stoddart S., Malone C. (eds), Temple Landscapes: Fragility, Change and Resilience of Holocene Environments in the Maltese Islands, McDonald Institute for Archaeological Research, Cambridge, 267-283. The Tobler function is rescaled by 0.25, i.e. the ratio between the average flock speed (1.5 km/h) and the maximum human walking speed (about 6.0 km/h) on favourable slopes. This function does not take the terrain factor N, since it models the walking pace of a flock rather than of a human walker.

Garmy, Kaddouri, Rozenblat, and Schneider's hiking function (`"gkrs"`):
$(4 * exp(-0.008 * ((atan(abs(x)) * 180/pi)^2))) * (1/N)$
slope in degrees; Garmy P., Kaddouri L., Rozenblat C., Schneider L. (2005), Logiques spatiales et "Systemes de villes" en Lodevois de l'Antiquite a la periode Moderne, in Berger J.-F., Bertoncello F., Braemer F., Davtian G., Gazenbeek M. (eds), Temps et espaces de l'homme en societe, analyses et modeles spatiaux en archeologie (XXVe rencontres internationales d'archeologie et d'histoire d'Antibes, 20-23 octobre 2004), Editions APDCA, Antibes, 335-346; as reported 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, 333-358.

Rees' hiking function (`"r"`):
$((1/(0.75 + 0.09 * x + 14.6 * x^2)) * 3.6) * (1/N)$
Rees W.G. (2004), Least-cost paths in mountainous terrain, Computers & Geosciences, 30(3), 203-209 (equation 1, with slope defined as height gained over horizontal distance, i.e. signed). Speed transposed from m/s to km/h. **Note**: movecost <= 3.1.0 applied the absolute value to the slope, so the linear term also slowed descents; fixed in 3.1.1 (the effect is small, as the linear coefficient is small).

Kondo-Seino's modified Tobler hiking function (`"ks"`):
$ifelse(x >= -0.07, (5.1 * exp(-2.25 * abs(x + 0.07))) * (1/N), (5.1 * exp(-1.5 * abs(x + 0.07))) * (1/N))$
Kondo Y., Seino Y. (2010), GPS-aided Walking Experiments and Data-driven Travel Cost Modeling on the Historical Road of Nakasendo-Kisoji (Central Highland Japan), in Frischer B., Webb Crawford J., Koller D. (eds), Making History Interactive, CAA Proceedings of the 37th International Conference (BAR International Series S2079), Archaeopress, Oxford, 158-165. **Note**: in movecost <= 2.x the steep-downhill branch could never be selected because the test was applied to the absolute slope; version 3.0 evaluates the signed slope as intended by Kondo and Seino, so walking speed on downhill gradients steeper than -7% now correctly uses the gentler decay coefficient (1.5). Results on such gradients therefore differ (correctly) from movecost <= 2.x.

Tripcevich's hiking function (`"trp"`):
$((4.028 * 46^2)/(((atan(x) * 180/pi) + 4.127)^2 + 46^2)) * (1/N)$
Tripcevich N. (2008), Estimating Llama Caravan Travel Speeds: Ethno-archaeological Fieldwork with a Peruvian Salt Caravan, poster, Archaeological Research Facility, University of California, Berkeley, https://escholarship.org/uc/item/79q8q6sn. An asymmetrical Cauchy function, h * w^2 / ((slope - x0)^2 + w^2), fitted to the mean speed (km/h) of each signed slope-integer

group, with slope in degrees (ATAN of elevation change over step length), h = 4.028, x0 = -4.127, w = 46.0; peak speed is therefore reached on a -4.1 degree descent. **Note**: movecost <= 3.1.0 applied the absolute value to the slope, which voided the asymmetry; fixed in 3.1.1.

Function for wheeled vehicles:

Wheeled-vehicle critical slope cost function (`"wcs"`):
$1/((1 + ((abs(x) * 100)/sl.crit)^2) * N)$
where `sl.crit` (critical slope, in percent) is "the transition where switchbacks become more effective than direct uphill or downhill paths", typically in the range 8-16; Herzog I. (2016), Potential and Limits of Optimal Path Analysis, in Bevan A., Lake M. (eds), Computational Approaches to Archaeological Spaces (pp. 179-211), Routledge.

Functions expressing abstract cost:

Relative energetic expenditure (`"ree"`):
$1/((tan((atan(abs(x)) * 180/pi) * pi/180)/tan(1 * pi/180)) * N)$
Conolly J., Lake M. (2006), Geographic Information Systems in Archaeology, Cambridge University Press, p. 220. The cost is the tangent of the slope relative to that of a 1-degree slope, so movement on level ground costs nothing (and the terrain factor has no effect there).

Bellavia's cost function (`"b"`):
$1/(((atan(abs(x)) * 180/pi) + 1) * N)$
slope in degrees; Bellavia G. (2002), Extracting 'natural pathways' from a digital elevation model. Applications to landscape archaeological studies, in Burenhult G., Arvidson J. (eds), Archaeological Informatics: Pushing the Envelope. CAA 2001, BAR International Series 1016, Archaeopress, Oxford, 5-12; as reported in Herzog I. (2020), cited above.

Eastman's cost function (`"e"`):
$1/((0.031 * (atan(abs(x)) * 180/pi)^2 - 0.025 * (atan(abs(x)) * 180/pi) + 1) * N)$
slope in degrees, taken as a magnitude (the source applies the equation to the values of a slope raster, which carries no direction of travel). The underlying cost equation (0.031 * p^2 - 0.025 * p + 1, with p the slope in degrees) was originally proposed by Eastman J.R. (1999), IDRISI32: Guide to GIS and Image Processing, Clark Labs, Clark University, Worcester (MA), p. 61. It was subsequently adopted, among others, by Barge O., Chataigner C. (2004), Un SIG pour l'analyse des approvisionnements: l'exemple de l'obsidienne en Armenie, ArcheoSciences, revue d'Archeometrie 28, 25-33, and reproduced by Vaissie E. (2021), Mobility of Paleolithic Populations: Biomechanical Considerations and Spatiotemporal Modelling, PaleoAnthropology 2021(1), 120-144 (see p. 124, with explicit reference to Eastman 1999: 61), the accessible source from which the equation was drawn into movecost.

Functions expressing cost as metabolic energy expenditure:

Pandolf et al.'s cost function (accumulated energy in kJ) (`"p"`):
$V/(1.5 * W + 2.0 * (W + L) * (L/W)^2 + N * (W + L) * (1.5 * (V^2) + 0.35 * V * (abs(x) * 100)))$
where W is the walker's body weight (kg), L the carried load (kg), V the velocity in m/s, N the

terrain coefficient (the η of the original publication, multiplying the velocity-dependent term once). If V is set to 0, it is worked out internally from the Tobler on-path function and varies with the slope. The bracketed expression M is a metabolic rate (W); dividing it by V gives the energy spent per metre (J/m), so that the accumulated value is energy (J, reported in kJ). As V is a speed along the ground, distances are planimetric unless `topo.dist = TRUE` in `mc_surface`, which understates energy slightly on steep terrain (about 4% at 30%). **Departure from the source**: the equation of Pandolf et al. (1977) was developed for level and uphill walking (Yokota et al. 2004, cited below) and, with a negative grade, turns negative on steep descents. movecost uses the absolute grade, i.e. the symmetric form reported by Herzog I. (2020, Table 18.2, cited below), which treats a descent as costly as an ascent of the same gradient; for a downhill-specific model use `"pcf"`. **Notes**: movecost <= 2.x additionally multiplied the whole expression by N, applying the terrain factor twice (fixed in 3.0); movecost <= 3.1.0 accumulated M * distance (W*m, labelled "Megawatts") rather than energy, so that with V = 0 the least-cost path did not minimise energy (fixed in 3.1.1). 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, doi:10.1152/jappl.1977.43.4.577.

Pandolf et al.'s cost function with downhill correction (accumulated energy in kJ) (`"pcf"`): $ifelse(x >= 0, V/M, V/(M - CF))$ where M is the Pandolf et al. metabolic rate above, computed with the signed grade G (percent), and $CF = N * ((G * (W + L) * V)/3.5 - ((W + L) * (G + 6)^2)/W + (25 - V^2))$, applied when G < 0, with the terrain factor multiplying the whole bracket; Yokota M., Berglund L.G., Santee W.R., Buller M.J., Hoyt R.W. (2004), Predicting individual physiological responses during marksmanship field training using an updated SCENARIO-J model, U.S. Army Research Institute of Environmental Medicine Technical Report T04-09, p. 4. Looney D.P. et al. (2019), Estimating energy expenditure during level, uphill, and downhill walking, Medicine & Science in Sports & Exercise 51(9), 1954-1960 (Table 2), print the correction with a signed grade too, but with a last term of + 25 * V^2 instead of (25 - V^2); movecost follows Yokota et al. (2004), the source of the correction as implemented in the SCENARIO-J model. As for `"p"`, the metabolic rate is divided by V so that the accumulated value is energy (kJ). Looney et al. (2019) report that the correction holds down to about -12% and overcorrects on steeper descents. Since CF is not zero at G = 0 (about -12 W with W = 70, L = 0, V = 1.2), the cost changes by a small step between level and slightly downhill movement; this comes from the source equation. **Notes**: in movecost <= 2.x the test selecting the downhill branch was applied to the absolute slope, so the downhill correction could never be selected (fixed in 3.0). Versions 3.0.0-3.1.0 then computed both M and CF with the absolute grade and applied the terrain factor to the first CF term only, so that descents cost more than ascents of the same gradient (e.g., with W = 70, L = 0, V = 1.2: 543 W instead of 195 W at -10%); fixed in 3.1.1.

Minetti et al.'s metabolic cost function (J/(kg*m); accumulated in J/kg) (`"m"`): $1/(((280.5*i^5) - (58.7*i^4) - (76.8*i^3) + (51.9*i^2) + (19.6*i) + 2.5)*N)$, with $i = max(x, -0.45)$ Minetti A.E., Moia C., Roi G.S., Susta D., Ferretti G. (2002), Energy cost of walking and running at extreme uphill and downhill slopes, Journal of Applied Physiology 93, 1039-1046. The polynomial is defined on the signed gradient, measured between -0.45 and +0.45; the measured minimum cost is on a descent of -0.10 (0.81 J/(kg*m)), and the polynomial's minimum on a descent of about -0.15. **Departure from the source**: below -0.45 the polynomial first decreases and then turns negative (about -0.56; see also Herzog 2016), which is physically meaningless; movecost therefore evaluates it at -0.45 for steeper descents, and `mc_surface` reports how many edges are affected. For terrain this steep the Herzog polynomial approximation (`"hrz"`) is preferable. **Note**: movecost <= 3.1.0 applied the absolute value to the gradient, which made descents as costly as ascents (e.g., 4.90

instead of 1.13 J/(kg*m) at -0.10); fixed in 3.1.1.

Herzog's metabolic cost function (J/(kg*m); accumulated in J/kg) (`"hrz"`):
$1/(((1337.8 * x^6) + (278.19 * x^5) - (517.39 * x^4) - (78.199 * x^3) + (93.419 * x^2) + (19.825 * x) + 1.64) * N)$
Herzog I. (2016), cited above, p. 5 ("s is mathematical slope"); also Herzog I. (2020, Table 18.2), cited above. 6th-degree polynomial approximation of Minetti et al.'s data on the signed slope, positive for all slopes, with its minimum on a descent of about -0.11. **Note**: movecost <= 3.1.0 applied the absolute value to the slope; fixed in 3.1.1.

Van Leusen's metabolic cost function (accumulated energy in kJ) (`"vl"`):
$V/(1.5*W+2.0*(W+L)*(L/W)^2+N*(W+L)*(1.5*(V^2)+0.35*V*abs(x*100+10)))$
Van Leusen P.M. (2002), Pattern to process: methodological investigations into the formation and interpretation of spatial patterns in archaeological landscapes, PhD thesis, University of Groningen, chapter 6. Van Leusen replaces the grade G of the Pandolf et al. equation by the absolute value of the signed grade shifted to the axis of cost symmetry, which he places at "-10% (or -6 degrees)" of slope. **Departure from the source**: the printed formula reads abs(G + 6), which mixes degrees with a grade in percent; movecost follows the stated axis and uses abs(G + 10), with G in percent, which also agrees with the minimum of the energy cost of walking near -10% (Minetti et al. 2002). As for `"p"`, the metabolic rate is divided by V so that the accumulated value is energy (kJ); the terrain factor is applied once. **Notes**: movecost <= 2.x applied the terrain factor twice (fixed in 3.0); movecost <= 3.1.0 used abs(G) + 10, which equals abs(G + 10) uphill but not downhill (e.g., 844 W instead of 256 W at -10% with W = 70, L = 0, V = 1.2), and accumulated W*m rather than energy; fixed in 3.1.1.

Llobera-Sluckin's metabolic cost function (kJ/m; accumulated in kJ) (`"ls"`):
$1/((2.635 + (17.37 * x) + (42.37 * x^2) - (21.43 * x^3) + (14.93 * x^4)) * N)$
Llobera M., Sluckin T.J. (2007), Zigzagging: Theoretical insights on climbing strategies, Journal of Theoretical Biology 249, 206-217 (equation 16, a quartic fit to Margaria's data on the signed slope; the authors stress that the metabolic curve is asymmetric). **Note**: movecost <= 3.1.0 applied the absolute value to the slope; fixed in 3.1.1.

Ardigo et al.'s metabolic cost function (J/(kg*m); accumulated in J/kg) (`"a"`):
$1/((1.866*exp(4.911*x)*V^2-3.773*exp(3.416*x)*V+(45.72*x^2+18.90*x)+4.456)*N)$
Ardigo L.P., Saibene F., Minetti A.E. (2003), The optimal locomotion on gradients: walking, running or cycling?, Eur J Appl Physiol 90, 365-371 (equation 1, with the gradient i signed, uphill and downhill). If V is set to 0, it is worked out internally from the Tobler on-path function. **Note**: movecost <= 3.1.0 applied the absolute value to the gradient and used 45.71 instead of the published 45.72; fixed in 3.1.1.

Hare's metabolic cost function (kcal/km; accumulated in kcal) (`"h"`):
$1/((48 + 30/(6 * exp(-3.5 * abs(x + 0.05)))) * N)$
Hare T.S. (2004), Using Measures of Cost Distance in the Estimation of Polity Boundaries in the Post Classic Yautepec valley, Mexico, Journal of Archaeological Science 31. The function gives kcal per km; the accumulated value is divided by 1000 so that it is expressed in kcal. **Note**: movecost <= 3.1.0 reported the accumulated sum without this division, under the label "cal/km"; fixed in 3.1.1.

Terrain factor (N):
virtually all the implemented cost functions can take into account a terrain factor (parameter `N`; 1 by default) representing the easiness/difficulty of moving on different terrain types. There are a few exceptions: the off-path functions (`"tofp"`, `"icmoffp"`, `"icfoffp"`), which already embed off-path walking (their on-path versions, `"t"`, `"icmonp"`, `"icfonp"`, do take N), and Alberti's function (`"alb"`), which models the walking pace of a flock rather than of a human walker. These functions ignore N, with a warning if a value other than 1 (or a terrain raster) is supplied.
The terrain factor can be a single value for the whole study area, or vary across it, via a raster of terrain factors or of terrain classes (see the `terrain` and `terrain.table` arguments of `mc_surface`).

The following reference list of terrain factors 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). It is divided into two sections: section (a) reports the terrain factors to be used for cost functions measuring time; section (b) for functions measuring cost other than time. The same values are returned as a data frame by `mc_terrain_factors`, ready for building the lookup table of a terrain-class raster.

(a) for time-based cost functions:

- 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) for cost functions measuring cost other than time:

- 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

Why speed-type and reciprocal-type functions are treated differently:
internally the package works with conductance (the reciprocal of cost), so walking-speed functions are used as they are (the accumulated value being the reciprocal of speed, i.e. pace, integrated over distance, which yields time), while the other functions are reciprocated so that the accumulated value corresponds to the original cost (see Nakoinz-Knitter (2016), "Modelling Human Behaviour in Landscapes", Springer, p. 183).

Value

A data frame with one row per cost function and columns:

- `code`: the identifier to pass as the `funct` parameter;
- `name`: extended name of the cost function;
- `type`: `"walking time"`, `"energy"`, `"abstract"`, or `"wheeled vehicle"`;
- `cost_unit`: unit of the accumulated cost (before any user-selected time conversion); for the energy functions this is the accumulated energy (kJ, J/kg, or kcal), not the per-metre unit given in the description of each function;
- `uses_N`, `uses_WLV`, `uses_slcrit`: whether the function responds to the terrain factor, to the W/L/V parameters, and to the critical-slope parameter respectively.

See Also

`mc_surface`

Examples

```
# list all the implemented cost functions
mc_cost_functions()
```

`mc_dtm` *Download elevation data for a study area*

Description

Acquires a Digital Terrain Model from online sources (via the *elevatr* package) for the area enclosed by the supplied polygon, and crops it to the polygon's extent. This replaces the implicit download mechanism of movecost <= 2.x's `studyplot` parameter (which remains available in `mc_surface` and simply calls this function).

Usage

```
mc_dtm(studyplot, z = 9)
```

Arguments

studyplot	polygon (sf, SpatVector, or legacy sp class) representing the study area for which elevation data are acquired.
z	zoom level of the downloaded elevation data (0 to 15; 9 by default).

Details

The zoom level z controls the resolution of the downloaded data (default 9, a trade-off between resolution and download time). For details on the data sources and the ground resolution per zoom level, see the *elevatr* documentation: https://cran.r-project.org/package=elevatr.

Note: the downloaded raster is returned in the CRS provided by the source. Before using it with mc_surface, make sure it is in a projected CRS (reproject with `terra::project()` if needed); a geographic (lon/lat) raster is rejected by `mc_surface()` with an informative error.

Value

A SpatRaster.

See Also

mc_surface

Examples

```
## Not run:
# elevation data for the area around Mt Etna (sample polygon in the package)
boundary <- mc_etna_boundary()
dtm <- mc_dtm(boundary, z = 9)

## End(Not run)
```

---

mc_export *Export movecost results to disk*

Description

Writes the spatial components of any movecost result object to disk: rasters as GeoTIFF, vector layers as GeoPackage (which, unlike the ESRI shapefiles produced by movecost <= 2.x, preserves full attribute names and supports mixed geometry without sidecar files). File names carry the cost function code as a suffix, as in movecost <= 2.x. For a cost surface built with a terrain raster (see mc_surface), the per-cell terrain factors actually used are also written (`terrain_N_<funct>.tif`).

Usage

```
mc_export(x, dir = ".", prefix = NULL, overwrite = FALSE)
```

Arguments

x	a movecost result object (any of movecost_surface, movecost_accum, movecost_paths, movecost_corridor, movecost_boundary, movecost_alloc, movecost_network, movecost_rank, movecost_comp).
dir	output directory (default: current working directory).
prefix	optional file-name prefix (default: the class-specific name).
overwrite	TRUE or FALSE (default): overwrite existing files.

Value

Invisibly, a character vector with the paths of the written files.

Examples

```
## Not run:
dtm <- mc_volc()
start <- mc_volc_loc(); destin <- mc_destin_loc()
surf <- mc_surface(dtm, funct = "t", move = 8)
acc <- mc_accum(surf, start, breaks = 0.05)
mc_export(acc, dir = tempdir())

## End(Not run)
```

---

mc_network *Least-cost network between multiple locations*

Description

Computes least-cost paths between multiple locations, either between all unique pairs (type = "allpairs") or from each location to its closest (cost-wise) neighbour (type = "neigh"), together with the full origin-to-origin cost matrix and, optionally, a path-density raster.

Usage

```
mc_network(
  surface,
  nodes,
  type = c("allpairs", "neigh"),
  density = FALSE,
  time = "h"
)
```

Arguments

surface	a movecost_surface object created by mc_surface.
nodes	locations to be connected (sf points, SpatVector, or legacy SpatialPointsDataFrame); at least two.
type	"allpairs" (default) for LCPs between all unique pairs of locations (in both directions, since costs are anisotropic), or "neigh" for LCPs from each location to its cost-wise nearest neighbour.
density	TRUE or FALSE (default): also compute the path-density rasters.
time	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).

Details

The graph stored in the mc_surface object is reused throughout: the cost matrix requires a single multi-source query and the paths one Dijkstra run per location, whereas movecost <= 2.x rebuilt the conductance matrix before any of this could start. For n locations this reduces the dominant cost from O(n) full matrix constructions to zero.

The path-density raster (density = TRUE) counts, for each cell, how many least-cost paths traverse it; the percentage version expresses the count against the total number of paths. Density is computed exactly, from the graph vertices of each path (no rasterisation approximation).

Value

An object of class movecost_network, a list with components

- paths: sf lines with from, to, cost, and length attributes;
- cost.matrix: full asymmetric cost matrix between locations (in the unit of the cost function; for time-based functions, in the unit selected via time);
- density.count, density.perc: path-density rasters (SpatRaster; NULL unless density = TRUE);
- nodes: the input locations (sf), with a node column holding the index (1..n) that matches the row and column names of cost.matrix and the node labels drawn by plot();
- metadata used by the plot method.

Plot it at any time with plot() (see plot.movecost_network).

See Also

mc_surface, plot.movecost_network

Examples

```
dtm <- mc_volc()
destin <- mc_destin_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)
nw <- mc_network(surf, nodes = destin, type = "allpairs")
```

```
nw$cost.matrix              # asymmetric origin-to-origin costs
plot(nw)

# network connecting each site to its cost-wise nearest neighbour
nw2 <- mc_network(surf, nodes = destin, type = "neigh")
plot(nw2)
```

`mc_paths` *Least-cost paths from an origin to one or more destinations*

Description

Computes least-cost path(s) (LCPs) from an origin to one or more destination locations, reusing the graph stored in a `mc_surface` object. Because terrain costs are anisotropic, the path back from a destination to the origin can differ from the outward path; set `return.base = TRUE` to also compute the return paths.

Usage

```
mc_paths(surface, origin, destin, return.base = FALSE, time = "h")
```

Arguments

`surface`	a `movecost_surface` object created by `mc_surface`.
`origin`	single origin location (sf points, SpatVector, or legacy SpatialPointsDataFrame); if more than one feature is supplied, the first is used.
`destin`	destination location(s) (same accepted classes).
`return.base`	TRUE or FALSE (default): also compute the least-cost paths from each destination back to the origin.
`time`	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).

Details

Each returned path carries its accumulated `cost` (in the unit of the surface's cost function; for time-based functions, in the unit selected via `time`) and its `length`. For time-based functions a `cost_hms` column additionally expresses the cost in sexagesimal format (HH:MM:SS). The same attributes are appended to the returned copy of the destination locations, mirroring the `dest.loc.w.cost` output of movecost <= 2.x.

Value

An object of class `movecost_paths`, a list with components

- `paths`: `sf` lines with `cost`, `length` and (for time-based functions) `cost_hms` attributes;
- `paths.back`: `sf` lines of the return paths (NULL unless `return.base = TRUE`);
- `destin`: copy of the destination locations with `cost` (and `cost_hms`) appended;
- `origin`: the origin (`sf`);
- metadata used by the plot method.

Plot it at any time with `plot()` (see `plot.movecost_paths`); no computation is re-run.

See Also

`mc_surface`, `plot.movecost_paths`, `mc_rank`, `mc_network`

Examples

```
dtm <- mc_volc()
start <- mc_volc_loc()
destin <- mc_destin_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)
lcp <- mc_paths(surf, origin = start, destin = destin)

lcp$paths          # sf object: costs, lengths, geometries
plot(lcp)

# paths back to the origin (anisotropy makes them differ):
lcp2 <- mc_paths(surf, start, destin[2, ], return.base = TRUE)
plot(lcp2)
```

`mc_rank` *Ranked (sub-optimal) least-cost paths between an origin and a destination*

Description

Computes the optimal least-cost path and up to `k - 1` progressively sub-optimal alternatives between two locations. Alternatives are obtained by iterative penalisation: after each path is found, the conductance of the graph edges incident to its cells is multiplied by `penalty` (0.01 by default), and Dijkstra's algorithm is run again. This discourages, without strictly forbidding, the reuse of earlier routes, and therefore returns spatially distinct alternatives rather than trivial one-cell deviations (which is what an exact k-shortest-path algorithm would return on a raster graph). The procedure is the same in spirit as movecost <= 2.x's `moverank()`, but operates directly on the graph edge weights, so the conductance matrix is never rebuilt and any number of ranks can be requested (movecost <= 2.x was limited to 6).

Usage

```
mc_rank(surface, origin, destin, k = 3, penalty = 0.01, time = "h")
```

Arguments

surface	a movecost_surface object created by mc_surface.
origin	single origin location (sf points, SpatVector, or legacy SpatialPointsDataFrame).
destin	single destination location (same accepted classes).
k	number of paths to return, optimal one included (default 3).
penalty	conductance multiplier applied to edges incident to already found paths (default 0.01; smaller values push alternatives further away from earlier routes).
time	unit for time-based cost functions: "h" (hours, default) or "m" (minutes).

Details

Each returned path carries its rank (1 = optimal), its accumulated cost computed on the *original* (unpenalised) graph - so costs of different ranks are directly comparable - and its length.

Value

An object of class movecost_rank, a list with components

- paths: sf lines with rank, cost, (for time-based functions) cost_hms, and length attributes;
- corridor: the least-cost corridor (SpatRaster) between the two locations, stored so that plot(x, type = "corridor") can draw the ranked paths over it without further computation;
- origin, destin: the input locations (sf);
- metadata used by the plot method.

Plot it at any time with plot(): plot(x) draws the paths over the terrain, plot(x, type = "corridor") draws them over the least-cost corridor, and plot(x, type = "chart") draws a bubble chart of length and cost by rank (see plot.movecost_rank).

See Also

mc_surface, plot.movecost_rank, mc_paths

Examples

```
dtm <- mc_volc()
start <- mc_volc_loc()
destin <- mc_destin_loc()

surf <- mc_surface(dtm, funct = "t", move = 8)
rk <- mc_rank(surf, origin = start, destin = destin[2, ], k = 3)

rk$paths        # ranks, costs, lengths
plot(rk)
```

mc_save — *Save and load movecost objects across R sessions*

Description

movecost result objects contain terra `SpatRaster` components, which hold pointers to external memory and therefore cannot be stored with plain `saveRDS()`/`readRDS()`. `mc_save()` wraps all raster components (via `terra::wrap()`) before serialising; `mc_load()` restores them. igraph components are serialised natively.

Usage

```
mc_save(x, file)

mc_load(file)
```

Arguments

`x`	a movecost result object.
`file`	path of the file to write/read (conventionally with extension `.rds`).

Value

`mc_save()` returns `file` invisibly; `mc_load()` returns the restored object.

Examples

```
## Not run:
surf <- mc_surface(mc_volc(), funct = "t", move = 8)
f <- tempfile(fileext = ".rds")
mc_save(surf, f)
surf2 <- mc_load(f)

## End(Not run)
```

mc_surface — *Create a reusable cost surface (the computational core of movecost 3.0)*

Description

`mc_surface()` builds, once, everything that any subsequent analysis needs: the directed graph of movement costs between adjacent DTM cells, derived from a user-selected cost function. The returned object is then fed to `mc_accum`, `mc_paths`, `mc_corridor`, `mc_boundary`, `mc_alloc`, `mc_network`, and `mc_rank`, none of which recomputes the graph.

Usage

```
mc_surface(
  dtm = NULL,
  funct = "t",
  move = 16,
  studyplot = NULL,
  z = 9,
  barrier = NULL,
  field = 0,
  cogn.slp = FALSE,
  topo.dist = FALSE,
  N = 1,
  sl.crit = 10,
  W = 70,
  L = 0,
  V = 1.2,
  terrain = NULL,
  terrain.table = NULL
)
```

Arguments

`dtm`	Digital Terrain Model: a terra `SpatRaster`, a legacy `RasterLayer`, or a file path readable by `terra::rast()`. If NULL, elevation data are downloaded for the area defined by `studyplot` via `mc_dtm` (requires the *elevatr* package).
`funct`	character: code of the cost function to be used (default `"t"`, Tobler's on-path hiking function). Run `mc_cost_functions()` for the complete annotated list.
`move`	number of directions in which cells are connected: 4 (rook's case), 8 (queen's case), or 16 (knight and one-cell queen moves; default).
`studyplot`	polygon (sf, SpatVector, or sp) defining the area for which online elevation data are downloaded when `dtm` is NULL.
`z`	zoom level (resolution) of the downloaded elevation data (0-15; default 9); only used when `dtm` is NULL. See `mc_dtm`.
`barrier`	lines or polygons (sf, SpatVector, or sp) representing areas where movement is inhibited (see Details).
`field`	conductance multiplier applied to edges touching the barrier (0 by default, i.e. movement inhibited).
`cogn.slp`	TRUE or FALSE (default): use the 'cognitive slope' in place of the real slope (see Details).
`topo.dist`	TRUE or FALSE (default): correct costs for topographic (surface) distance rather than planar distance (see Details).
`N`	terrain factor representing ease of movement (1 by default; see `mc_cost_functions` for reference values): a single positive number, applied to the whole study area. If `terrain` is supplied, `N` is used only for the cells with no terrain value (see Details).

`sl.crit`	critical slope in percent (default 10), used only by the wheeled-vehicle cost function.
`W`	walker's body weight in kg (default 70), used by the Pandolf and Van Leusen functions.
`L`	carried load in kg (default 0), used by the Pandolf and Van Leusen functions.
`V`	walking speed in m/s (default 1.2), used by the Pandolf, Van Leusen, and Ardigo functions; if 0, speed is derived cell-by-cell from the Tobler on-path function.
`terrain`	optional raster of terrain factors or terrain classes (SpatRaster, RasterLayer, or a file path readable by `terra::rast()`), making the terrain factor vary across the study area (see Details). NULL (default) applies the scalar `N` everywhere.
`terrain.table`	optional data frame translating the classes of a categorical `terrain` raster into terrain factors, with columns `class` (numeric codes, or the category labels of a terra factor raster) and `N` (terrain factor); a two-column data frame without these names is read as class, then N. Required for a terra factor raster; if NULL, the cell values of a numeric raster are taken as terrain factors.

Details

Why this design: in movecost <= 2.x every analysis function rebuilt the conductance matrix from scratch at each call (a network between n sites implied n or more redundant constructions). Building the graph once and reusing it makes multi-analysis workflows and large location sets dramatically faster, and makes the package's internal structure modular: one function, one responsibility.

Cost model: for each pair of adjacent cells (4, 8, or 16 neighbours, set via `move`), the terrain slope is computed as rise over run between cell centres, signed in the direction of travel (positive uphill, negative downhill), so movement costs are anisotropic. The user-selected cost function (see `mc_cost_functions` for all 26 implemented functions, their formulas, units, and references) maps the slope to either a walking speed or the reciprocal of a per-metre cost; the edge cost is then the distance between cell centres divided by speed (yielding seconds) or multiplied by the per-metre cost. Accumulated costs and least-cost paths are obtained with Dijkstra's algorithm via the *igraph* library.

Cognitive slope: if `cogn.slp = TRUE`, positive slopes are multiplied by 1.99 and negative slopes by 2.31 before the cost function is applied, following Pingel T.J. (2013), Modeling Slope as a Contributor to Route Selection in Mountainous Areas, Cartography and Geographic Information Science, 37(2), 137-148.

Topographic (surface) distance: if `topo.dist = TRUE`, costs are corrected by the factor $sqrt(1 + slope^2)$, accounting for the actual distance travelled over sloped terrain rather than its planar projection. The correction is clearly appropriate for time-based functions; for metabolic functions, which were empirically derived against planar distance, it should be used with caution. In gently undulating landscapes the difference is negligible; it matters in rugged terrain with slopes beyond 20-30 percent.

Barriers: areas where movement is inhibited can be supplied via `barrier` (any sf, SpatVector, or legacy sp lines/polygons object). Edges incident to cells touched by the barrier have their conductance multiplied by `field`: 0 (default) makes the barrier impassable, while intermediate values

(e.g. 0.01) penalise but do not forbid crossing. Note that with `move = 16`, knight-move connections can "jump" a thin linear barrier, exactly as in movecost <= 2.x; use `move = 8` if that must not happen.

Terrain factor raster: the terrain factor N (see `mc_cost_functions`) can vary across the study area. Supply a raster via `terrain` in either of two forms:

- a *numeric* raster whose cells hold terrain factors directly (e.g. 1.00 on roads, 1.79 on bogs);
- a *categorical* raster of terrain classes (integer codes, or a terra factor raster with category labels), together with a lookup table `terrain.table` pairing each class with its terrain factor. `mc_terrain_factors` returns documented reference values from which the table can be built.

The scalar `N` keeps its role and acts as the *fallback* value: cells with no terrain value (NA in the raster, cells outside its extent, or classes missing from the table) take `N`, so a raster can map only the terrain types of interest (roads, bogs, ...) and leave the rest to a single general value (e.g. `N = 1.67`, off-path). A message reports how many cells were filled in this way; a warning lists classes missing from the table. Terrain factors must be positive; `Inf` makes a cell impassable.
Each edge between two adjacent cells takes the *mean* of their terrain factors. Half of an edge lies in each cell, and for every implemented function the edge cost is linear in N (affine for the Pandolf and Van Leusen functions, where N multiplies only the terrain-dependent term of the metabolic equation); the mean therefore reproduces exactly the cost of crossing half of the edge on each terrain. Note that N is applied *inside* the cost function, as with the scalar N, and not as a multiplier of the final cost: for the Pandolf-type functions the two are different. With `move = 16`, knight-move edges pass close to two further cells whose terrain is not considered (the same simplification made for elevation).
If the raster's coordinate system or grid differs from the DTM's, it is aligned to the DTM with nearest-neighbour resampling (`terra::project(method = "near")`), which never invents values and is therefore safe for class codes; a warning is issued, as the data are changed. For full control, align the raster beforehand.
A few cost functions do not take a terrain factor (see the `uses_N` column of `mc_cost_functions()`) and ignore both `N` and `terrain`, with a warning: the off-path functions (`"tofp"`, `"icmoffp"`, `"icfoffp"`), which already embed off-path walking, and Alberti's function (`"alb"`), which models the walking pace of a flock rather than of a human walker. To model terrain with a raster, use the on-path version of an off-path function (`"t"`, `"icmonp"`, `"icfonp"`), so that the terrain effect is not counted twice.
A caution: published terrain factors were obtained in different studies, under specific conditions and for specific cost models (see `mc_terrain_factors`); a terrain raster makes their use easy, but does not make them more accurate.

Irregular DTMs: cells with NoData (for instance, sea around a coastline) are simply excluded from the graph, so accumulated costs and least-cost paths can never traverse them. The `irregular.dtm` workaround of movecost <= 2.x is therefore no longer needed and has no equivalent here.

Coordinate system: the DTM must use a projected coordinate system with planar units in metres; geographic (lon/lat) rasters are rejected with an informative error.

Value

An object of class `movecost_surface`: a list with components

- `dtm`: the input DTM (`SpatRaster`);
- `graph`: the directed igraph cost graph (one vertex per cell);
- `cost.raster`: `SpatRaster` storing, for each cell, the mean value of the cost function over the edges leaving the cell (walking speed in km/h for speed-based functions, conductance otherwise) - the analogue of the `cost.surface` component of movecost <= 2.x;
- `funct`, `params`, `move`: metadata describing how the graph was built (used by downstream functions and plot methods);
- `terrain`: `SpatRaster` of the terrain factor actually used for each cell (after alignment, class lookup, and filling with `N`), or NULL if no terrain raster was supplied; draw it with `plot(x, type = "terrain")`.

The object has `print()` and `plot()` methods; `plot()` returns a ggplot2 object (see `plot.movecost_surface`).

Saving to disk: because SpatRaster objects hold external pointers, use `mc_save` and `mc_load` (not bare `saveRDS()`) to store movecost objects across sessions.

References

Alberti (2019) <doi:10.1016/j.softx.2019.100331>.

See Also

`mc_cost_functions`, `mc_accum`, `mc_paths`, `mc_corridor`, `mc_boundary`, `mc_alloc`, `mc_network`, `mc_rank`, `mc_dtm`, `mc_terrain_factors`, `mc_save`; visualisation: `plot.movecost_surface`

Examples

```
# load the package sample DTM (Maungawhau volcano) and locations
dtm <- mc_volc()
start <- mc_volc_loc()
destin <- mc_destin_loc()

# build the cost surface once (Tobler's on-path function, 8 directions)
surf <- mc_surface(dtm, funct = "t", move = 8)

# ...and reuse it across analyses without recomputation:
acc <- mc_accum(surf, origin = start, breaks = 0.05)
lcp <- mc_paths(surf, origin = start, destin = destin)

# visualise on demand (no re-running of any computation):
plot(acc)
plot(lcp)

# a terrain factor varying across the area: a (fictitious) categorical
# raster with class 1 below 120 m and class 2 (rock) above, translated
# into terrain factors via a lookup table
terr <- terra::classify(dtm, rbind(c(-Inf, 120, 1), c(120, Inf, 2)))
tf <- mc_terrain_factors("time")
tab <- data.frame(class = 1:2,
                  N = tf$N[match(c("Lawn grass", "Rock"), tf$terrain)])
```

```
surf_t <- mc_surface(dtm, funct = "t", move = 8,
                     terrain = terr, terrain.table = tab)
surf_t
plot(surf_t, type = "terrain")
```

`mc_terrain_factors` *Reference terrain factors (N) for use with a terrain raster*

Description

Returns, as a data frame, the reference list of terrain factors reported in the help page of `mc_cost_functions`. It is meant as a starting point for building the lookup table (`terrain.table`) that `mc_surface` uses to translate a categorical terrain (land-cover) raster into terrain factors: pick the rows that describe the terrain types of the study area, and pair them with the class codes (or labels) of the raster (see Examples).

Usage

```
mc_terrain_factors(type = c("time", "other", "all"))
```

Arguments

`type`	character: `"time"` (default) for time-based cost functions, `"other"` for functions measuring cost other than time, or `"all"` for both sets.

Details

The values are taken from 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). They are split into two sets: `type = "time"` for cost functions measuring walking time, and `type = "other"` for functions measuring cost other than time (energy, abstract cost). For ploughed fields, for which the source gives two values (1.30 or 1.50), both are returned as separate rows.

A note of caution: these factors were obtained in different studies, under specific conditions, and for specific cost models. They are convenient, documented defaults, not calibrated constants; when better local evidence is available, it should be preferred.

Value

A data frame with columns `terrain` (description of the terrain type), `N` (terrain factor), and `type` (`"time"` or `"other"`).

See Also

`mc_surface` (arguments `terrain` and `terrain.table`), `mc_cost_functions`

Examples

```
# reference values for time-based cost functions
tf <- mc_terrain_factors("time")
tf

# build a lookup table for a raster whose cells are coded 1, 2, 3
# (1 = lawn grass, 2 = bog, 3 = rock)
tab <- data.frame(class = 1:3,
                  N = tf$N[match(c("Lawn grass", "Bog", "Rock"), tf$terrain)])
tab
```

---

movecost-data *Sample data bundled with movecost*

Description

movecost 3.0 ships its sample datasets as standard GIS files (GeoTIFF and GeoPackage) read on demand by the accessor functions documented here. This frees the package data from any dependency on the retiring *raster*/*sp* stack used by movecost <= 2.x (whose .rda data files could not be loaded without those packages installed).

Usage

```
mc_volc()

mc_malta_dtm()

mc_volc_loc()

mc_destin_loc()

mc_springs()

mc_etna_start()

mc_etna_end()

mc_etna_boundary()
```

Details

Available accessors:

- `mc_volc()`: sample DTM (a volcano, Maungawhau/Mt Eden area; projected CRS, about 10 m resolution); `SpatRaster`.

- `mc_malta_dtm()`: DTM of Malta (40 m resolution); its coastline NoData margin makes it the reference dataset for analyses on irregular DTMs; `SpatRaster`.
- `mc_volc_loc()`: one start location on the volc DTM; `sf`.
- `mc_destin_loc()`: nine destination locations on the volc DTM; `sf`.
- `mc_springs()`: 49 springs on the Malta DTM; `sf`.
- `mc_etna_start()`, `mc_etna_end()`: start and end locations in the Mt Etna area (Sicily), for use with online elevation download (`mc_dtm`); `sf`.
- `mc_etna_boundary()`: study-area polygon for the Mt Etna area; `sf`.

Note on irregular DTMs: in movecost 3.0, NoData cells are excluded from the cost graph automatically, so least-cost paths on `mc_malta_dtm()` follow the coastline without any special setting (the `irregular.dtm` workaround of movecost <= 2.x is no longer needed).

Note on coordinate reference systems: the movecost <= 2.x datasets carried legacy proj4 definitions without authority codes; for version 3.0 they have been assigned the corresponding EPSG codes, which survive file round-trips identically across formats: volc and its locations EPSG:27200 (NZGD49 / New Zealand Map Grid), the Malta DTM and springs EPSG:23033 (ED50 / UTM 33N, matching the original International-ellipsoid UTM definition), the Etna datasets EPSG:23032 (ED50 / UTM 32N, as in the original definition). Coordinates are unchanged.

Value

`mc_volc()` and `mc_malta_dtm()` return a `SpatRaster`; all other accessors return an `sf` object.

See Also

`mc_surface`, `mc_dtm`

Examples

```
dtm <- mc_volc()
origin <- mc_volc_loc()
destinations <- mc_destin_loc()


# the Malta DTM: paths cannot cross the sea (NoData cells)
malta <- mc_malta_dtm()
springs <- mc_springs()
surf <- mc_surface(malta, funct = "t", move = 8)
lcp <- mc_paths(surf, origin = springs[5, ], destin = springs[15, ])
plot(lcp)
```

`plot.movecost_accum` *Plot an accumulated cost surface*

Description

Returns a ggplot object showing the accumulated cost surface with isolines and origin location(s), or the isolines alone (`type = "contour"`), reproducing the two output types of movecost <= 2.x's `outp` parameter - but on demand, without re-running the analysis.

Usage

```
## S3 method for class 'movecost_accum'
plot(
  x,
  type = c("raster", "contour"),
  cont.lab = TRUE,
  transp = 0.5,
  cex.breaks = 2.5,
  iso.col = "white",
  iso.lwd = 0.3,
  ...
)
```

Arguments

`x`	a `movecost_accum` object.
`type`	"raster" (default; accumulated surface, slope-shade, isolines) or "contour" (isolines only).
`cont.lab`	TRUE (default) or FALSE: label the isolines.
`transp`	transparency of the slope-shade overlay (0.5 by default).
`cex.breaks`	size of the isoline labels (2.5 by default, in ggplot2 text-size units).
`iso.col`	colour of the isolines and their labels (default `"white"`, which reads clearly against the dark low-cost core of the viridis "C" palette; over the `"contour"` output, where there is no coloured background, `"grey20"` is used automatically unless the user overrides `iso.col`).
`iso.lwd`	line width of the isolines (0.3 by default).
`...`	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

`mc_accum`

`plot.movecost_alloc` *Plot cost-allocation zones*

Description

Returns a ggplot object showing the cost-allocation zones (one colour per origin), with the origins and, optionally, the isolines of the minimum accumulated cost surface.

Usage

```
## S3 method for class 'movecost_alloc'
plot(
  x,
  isolines = FALSE,
  transp = 0.5,
  iso.col = "white",
  iso.lwd = 0.3,
  origin.lab = TRUE,
  cex.origin.lab = 3,
  ...
)
```

Arguments

`x`	a `movecost_alloc` object.
`isolines`	TRUE or FALSE (default): overlay the cost isolines.
`transp`	transparency of the slope-shade overlay (0.5 by default).
`iso.col`	colour of the isolines (default `"white"`, which reads clearly against the coloured allocation zones).
`iso.lwd`	line width of the isolines (0.3 by default).
`origin.lab`	TRUE (default) or FALSE: label each origin with its index. The indices match the zone identifiers (the cell values of the allocation raster and the `origin_id` of `zones`), so each origin can be related to the zone allocated to it.
`cex.origin.lab`	size of the origin labels (3 by default, in ggplot2 text-size units).
`...`	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

`mc_alloc`

plot.movecost_boundary *Plot cost-limit boundaries*

Description

Returns a ggplot object showing the DTM with the cost-limit boundary polygon of each origin.

Usage

```
## S3 method for class 'movecost_boundary'
plot(x, transp = 0.5, ...)
```

Arguments

x	a movecost_boundary object.
transp	transparency of the slope-shade overlay (0.5 by default).
...	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

mc_boundary

plot.movecost_comp *Plot a cost-function comparison*

Description

Returns a ggplot object showing either the least-cost paths produced by the compared cost functions, colour-coded by function (type = "paths", default), or a chart comparing the path length and the accumulated cost across functions (type = "chart").

Usage

```
## S3 method for class 'movecost_comp'
plot(x, type = c("paths", "chart"), transp = 0.5, ...)
```

Arguments

x	a movecost_comp object.
type	"paths" (default) or "chart".
transp	transparency of the slope-shade overlay (0.5 by default).
...	unused.

Details

For the chart, when several destinations are compared (three or more) the panels are **boxplots** showing, for each cost function, the distribution across destinations of the path length and of the cost, reproducing the add.chart output of movecost <= 2.x's movecomp(). With fewer than three destinations a boxplot would be degenerate, so bars are drawn instead. The cost panel is split by cost unit, so cost is only ever compared between functions that share a unit (length, in metres, is always directly comparable). When the comparison was computed with return.base = TRUE, the outward and the return paths are shown in separate rows ("outward" and "return"), as movecost <= 2.x did with a separate set of boxplots for the paths back to the origin.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

mc_comp

---

plot.movecost_corridor

Plot a least-cost corridor

Description

Returns a ggplot object showing the corridor surface (low values = cheap movement) with the input locations. For the two-location case computed with lcp = TRUE, the least-cost paths A-to-B and B-to-A are overlaid (they differ under anisotropic cost functions).

Usage

```
## S3 method for class 'movecost_corridor'
plot(x, transp = 0.5, lcp.lwd = 0.4, ...)
```

Arguments

x	a movecost_corridor object.
transp	transparency of the slope-shade overlay (0.5 by default).
lcp.lwd	line width of the overlaid least-cost paths (0.4 by default).
...	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

mc_corridor

`plot.movecost_network` *Plot a least-cost network*

Description

Returns a ggplot object showing either the network of least-cost paths over the DTM (`type = "paths"`, default) or the path-density raster (`type = "density"`, available if the network was computed with `density = TRUE`).

Usage

```
## S3 method for class 'movecost_network'
plot(
  x,
  type = c("paths", "density"),
  transp = 0.5,
  lcp.lwd = 0.3,
  node.lab = TRUE,
  cex.node.lab = 3,
  ...
)
```

Arguments

`x`	a `movecost_network` object.
`type`	"paths" (default) or "density".
`transp`	transparency of the slope-shade overlay (0.5 by default).
`lcp.lwd`	line width of the network paths (0.3 by default).
`node.lab`	TRUE (default) or FALSE: label each node with its index. The indices match the row and column names of the returned `cost.matrix`, so a node on the map can be related to its entries in the matrix (reproducing the behaviour of movecost <= 2.x).
`cex.node.lab`	size of the node labels (3 by default, in ggplot2 text-size units).
`...`	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

mc_network

`plot.movecost_paths` *Plot least-cost paths*

Description

Returns a ggplot object showing the DTM (terrain colours plus slope-shade), the least-cost path(s), the origin (black dot) and destination(s) (red dots), optionally labelled with the accumulated cost, and the return paths (dashed) if they were computed.

Usage

```
## S3 method for class 'movecost_paths'
plot(
  x,
  destin.lab = TRUE,
  cex.lcp.lab = 2.5,
  transp = 0.5,
  lcp.lwd = 0.3,
  plot.barrier = FALSE,
  ...
)
```

Arguments

`x`	a `movecost_paths` object.
`destin.lab`	TRUE (default) or FALSE: label each destination with its accumulated cost.
`cex.lcp.lab`	size of the destination labels (2.5 by default).
`transp`	transparency of the slope-shade overlay (0.5 by default).
`lcp.lwd`	line width of the least-cost path(s) (0.3 by default; the return paths, if any, are drawn at 80% of this width).
`plot.barrier`	TRUE or FALSE (default): draw the barrier (blue) if one was used when building the surface.
`...`	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

mc_paths

plot.movecost_rank *Plot ranked least-cost paths*

Description

Returns a ggplot object showing, depending on `type`, the ranked paths over the terrain (`"map"`, default), the same paths drawn over the least-cost corridor between the two locations (`"corridor"`, the equivalent of movecost <= 2.x's `use.corr`), or a bubble chart of path length against rank with bubble size proportional to cost (`"chart"`, the equivalent of movecost <= 2.x's `add.chart`). No manual data assembly is required for any of them.

Usage

```
## S3 method for class 'movecost_rank'
plot(x, type = c("map", "corridor", "chart"), transp = 0.5, lcp.lwd = 0.6, ...)
```

Arguments

`x`	a `movecost_rank` object.
`type`	`"map"` (default), `"corridor"`, or `"chart"` (see Details).
`transp`	transparency of the slope-shade overlay (0.5 by default).
`lcp.lwd`	line width of the ranked paths (0.6 by default).
`...`	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

`mc_rank`

plot.movecost_surface *Plot a movecost cost surface*

Description

Returns a ggplot object showing the per-cell cost raster (walking speed in km/h for speed-based functions, conductance otherwise) with a translucent slope-shade overlay. If the surface was built with a terrain raster (see the `terrain` argument of `mc_surface`), `type = "terrain"` shows the terrain factor (N) of each cell instead, as actually used to build the surface (that is, after alignment to the DTM grid, class lookup, and filling of cells with no terrain value).

Usage

```
## S3 method for class 'movecost_surface'
plot(x, transp = 0.5, type = c("cost", "terrain"), ...)
```

Arguments

x	a movecost_surface object.
transp	transparency of the slope-shade overlay (0.5 by default).
type	"cost" (default) for the per-cell cost raster, or "terrain" for the terrain factor of each cell (only for surfaces built with a terrain raster).
...	unused.

Value

A ggplot object (draw it by printing; customise it with +).

See Also

mc_surface

Index

mc_accum, 2, *25*, *29*, *33*
mc_alloc, 4, *25*, *29*, *34*
mc_boundary, 5, *25*, *29*, *35*
mc_comp, 6, *36*
mc_corridor, 8, *25*, *29*, *37*
mc_cost_functions, *3*, *7*, 11, *26–30*
mc_destin_loc *(*movecost-data*)*, 31
mc_dtm, 18, *26*, *29*, *32*
mc_etna_boundary *(*movecost-data*)*, 31
mc_etna_end *(*movecost-data*)*, 31
mc_etna_start *(*movecost-data*)*, 31
mc_export, *3*, 19
mc_load, *29*
mc_load *(*mc_save*)*, 25
mc_malta_dtm *(*movecost-data*)*, 31
mc_network, 20, *23*, *25*, *29*, *38*
mc_paths, *7*, *10*, 22, *24, 25*, *29*, *38*
mc_rank, 23, *23*, *25*, *29*, *39*
mc_save, 25, *29*
mc_springs *(*movecost-data*)*, 31
mc_surface, *2–8*, *10*, *15*, *17–19*, *21–24*, 25, *30*, *32*, *39, 40*
mc_terrain_factors, *17*, *28, 29*, 30
mc_volc *(*movecost-data*)*, 31
mc_volc_loc *(*movecost-data*)*, 31
movecost-data, 31

plot.movecost_accum, *3*, 33
plot.movecost_alloc, *4, 5*, 34
plot.movecost_boundary, *6*, 35
plot.movecost_comp, *7*, 35
plot.movecost_corridor, *10*, 36
plot.movecost_network, *21*, 37
plot.movecost_paths, *23*, 38
plot.movecost_rank, *24*, 39
plot.movecost_surface, *29*, 39