Package: windfarmGA (via r-universe)

January 21, 2025

Title Genetic Algorithm for Wind Farm Layout Optimization

Version 4.0.0.9000

Maintainer Sebastian Gatscha <sebastian_gatscha@gmx.at>

Description The genetic algorithm is designed to optimize wind farms of any shape. It requires a predefined amount of turbines, a unified rotor radius and an average wind speed value for each incoming wind direction. A terrain effect model can be included that downloads an 'SRTM' elevation model and loads a Corine Land Cover raster to approximate surface roughness.

Depends R (>= 4.1.0)

Imports Rcpp, terra, sf, RColorBrewer, calibrate, grDevices, graphics, magrittr, methods, stats, utils

LinkingTo Rcpp

LazyData TRUE

License MIT + file LICENSE

URL https://ysosirius.github.io/windfarmGA/index.html

BugReports https://github.com/YsoSirius/windfarmGA/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), foreach, parallel, doParallel, progress, stars, raster, leaflet, elevatr (>= 0.99.0), ggplot2, gstat, rworldmap

X-schema.org-keywords windfarm-layout, optimization, genetic-algorithm, renewable-energy, r, rstats, r-package

Config/testthat/edition 3

Config/testthat/parallel true

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

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

RemoteUrl https://github.com/ysosirius/windfarmga

RemoteRef HEAD

RemoteSha 577a9276ebc381afc80148d8bb8fd1c213b6ab99

Contents

`windfarmGA-package` *windfarmGA: Genetic Algorithm for Wind Farm Layout Optimization*

Description

The genetic algorithm is designed to optimize wind farms of any shape. It requires a predefined amount of turbines, a unified rotor radius and an average wind speed value for each incoming wind direction. A terrain effect model can be included that downloads an 'SRTM' elevation model and loads a Corine Land Cover raster to approximate surface roughness.

Details

[Stable]


A package to optimize small wind farms with irregular shapes using a genetic algorithm. It requires a fixed amount of turbines, a fixed rotor radius and an average wind speed value for each incoming wind direction. A terrain effect model can be included which downloads a digital elevation model and a Corine Land Cover raster to approximate surface roughness. Further information can be found at the description of the function `genetic_algorithm`.

Author(s)

Maintainer: Sebastian Gatscha `<sebastian_gatscha@gmx.at>`

See Also

Useful links:

- Documentation Github.io
- Documentation
- Master Thesis
- Shiny App
- Report Issues

`barometric_height` *Calculates Air Density, Air Pressure and Temperature according to the Barometric Height Formula*

Description

Calculates air density, temperature and air pressure respective to certain heights according to the International standard atmosphere and the barometric height formula.

Usage

```
barometric_height(data, height, po = 101325, ro = 1.225)
```

Arguments

`data`	A data.frame containing the height values
`height`	Column name of the height values
`po`	Standardized air pressure at sea level (101325 Pa)
`ro`	Standardized air density at sea level (1,225 kg per m3)

Value

Returns a data.frame with height values and corresponding air pressures, air densities and temperatures in Kelvin and Celsius.

See Also

Other Wind Energy Calculation Functions: `calculate_energy()`, `circle_intersection()`, `get_dist_angles()`, `turbine_influences()`

Examples

```
data <- matrix(seq(0, 5000, 500))
barometric_height(data)
plot.ts(barometric_height(data))
```

big_shape *A POLYGON with an area of ~70 km2*

Description

A POLYGON with an area of ~70 km2

Usage

```
big_shape
```

Format

An object of class `sf` (inherits from `data.frame`) with 1 rows and 1 columns.

calculate_energy *Calculate Energy Outputs of Individuals*

Description

Calculate the energy output and efficiency rates of an individual in the current population under all given wind directions and speeds. If the terrain effect model is activated, the main calculations to model those effects will be done in this function.

Usage

```
calculate_energy(
  sel,
  referenceHeight,
  RotorHeight,
  SurfaceRoughness,
  wnkl,
  distanz,
  polygon1,
  RotorR,
  dirSpeed,
  srtm_crop,
  topograp,
  cclRaster,
  weibull,
  plotit = FALSE
)
```

Arguments

sel	A matrix of an individual of the current population
referenceHeight	The height at which the incoming wind speeds were measured. Default is RotorHeight
RotorHeight	The height of the turbine hub
SurfaceRoughness	A surface roughness length in meters. With the terrain effect model, a surface roughness is calculated for every grid cell using the elevation and land cover data. Default is 0.3
wnkl	The angle from which wake influences are considered to be negligible
distanz	The distance after which wake effects are considered to be eliminated
polygon1	The considered area as Simple Feature Polygon
RotorR	The desired rotor radius in meter
dirSpeed	The wind speed and direction data.frame
srtm_crop	The first element of the terrain_model resulting list
topograp	Boolean value, which indicates if the terrain effect model should be enabled or not. Default is FALSE
cclRaster	The second element of the terrain_model resulting list
weibull	A boolean value that specifies whether to take Weibull parameters into account. If TRUE, the wind speed values of vdirspe are ignored. The algorithm will calculate the mean wind speed for every wind turbine according to the Weibull parameters. Default is FALSE
plotit	If TRUE, the process will be plotted. Default is FALSE

Value

Returns a list of an individual of the current generation with resulting wake effects, energy outputs, efficiency rates for every wind direction. The length of the list corresponds to the number of different wind directions.

See Also

Other Wind Energy Calculation Functions: barometric_height(), circle_intersection(), get_dist_angles(), turbine_influences()

Examples

```
## Create a random Polygon
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
```

```
  crs = 3035
))

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plot_windrose(
  data = data.in, spd = data.in$ws,
  dir = data.in$wd, dirres = 10, spdmax = 20
)

## Assign the rotor radius and a factor of the radius for grid spacing.
Rotor <- 50
fcrR <- 3
resGrid <- grid_area(
  shape = Polygon1, size = Rotor * fcrR, prop = 1,
  plotGrid = TRUE
)
## Assign the indexed data frame to new variable. Element 2 of the list
## is the grid, saved as Simple Feature Polygons.
resGrid1 <- resGrid[[1]]

## Create an initial population with the indexed Grid, 15 turbines and
## 100 individuals.
initpop <- init_population(Grid = resGrid1, n = 15, nStart = 100)

## Calculate the expected energy output of the first individual of the
## population.
par(mfrow = c(1, 2))
plot(Polygon1)
points(initpop[[1]][, "X"], initpop[[1]][, "Y"], pch = 20, cex = 2)
plot(resGrid[[2]], add = TRUE)
resCalcEn <- calculate_energy(
  sel = initpop[[1]], referenceHeight = 50,
  RotorHeight = 50, SurfaceRoughness = 0.14, wnkl = 20,
  distanz = 100000, dirSpeed = data.in,
  RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
  weibull = FALSE
)
resCalcEn <- as.data.frame(resCalcEn)
plot(Polygon1, main = resCalcEn[, "Energy_Output_Red"][[1]])
points(x = resCalcEn[, "Bx"], y = resCalcEn[, "By"], pch = 20)


## Create a variable and multidirectional wind data.frame and plot the
## resulting wind rose
data.in10 <- data.frame(ws = runif(10, 1, 25), wd = runif(10, 0, 360))
windrosePlot <- plot_windrose(
  data = data.in10, spd = data.in10$ws,
  dir = data.in10$wd, dirres = 10, spdmax = 20
)

## Calculate the energy outputs for the first individual with more than one
```

```
## wind direction.
resCalcEn <- calculate_energy(
  sel = initpop[[1]], referenceHeight = 50,
  RotorHeight = 50, SurfaceRoughness = 0.14, wnkl = 20,
  distanz = 100000, dirSpeed = data.in10,
  RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
  weibull = FALSE
)
```

`circle_intersection` *Get area of intersecting circles*

Description

Calculate the intersection area of two circles with different radii and different heights

Usage

```
circle_intersection(r1, r2, h1, h2, dx)
```

Arguments

`r1`	The radius of circle 1
`r2`	The radius of circle 2
`h1`	The height of the circle center 1
`h2`	The height of the circle center 2
`dx`	The distance on the x-axis between both centers

Value

A numeric value

See Also

Other Wind Energy Calculation Functions: `barometric_height()`, `calculate_energy()`, `get_dist_angles()`, `turbine_influences()`

crossover *Crossover Method*

Description

The crossover method creates new offspring with the selected individuals by permutating their genetic codes.

Usage

```
crossover(se6, u, uplimit, crossPart = c("EQU", "RAN"), verbose, seed)
```

Arguments

se6	The selected individuals. The output of selection
u	The crossover point rate
uplimit	The upper limit of allowed permutations
crossPart	The crossover method. Either "EQU" or "RAN"
verbose	If TRUE, will print out further information
seed	Set a seed for comparability. Default is NULL

Value

Returns a binary coded matrix of all permutations and all grid cells, where 0 indicates no turbine and 1 indicates a turbine in the grid cell.

See Also

Other Genetic Algorithm Functions: fitness(), genetic_algorithm(), init_population(), mutation(), selection(), trimton()

Examples

```
## Create two random parents with an index and random binary values
Parents <- data.frame(
  ID = 1:20,
  bin = sample(c(0, 1), 20, replace = TRUE, prob = c(70, 30)),
  bin.1 = sample(c(0, 1), 20, replace = TRUE, prob = c(30, 70))
)

## Create random Fitness values for both individuals
FitParents <- data.frame(ID = 1, Fitness = 1000, Fitness.1 = 20)

## Assign both values to a list
CrossSampl <- list(Parents, FitParents)
## Cross their data at equal locations with 2 crossover parts
crossover(CrossSampl, u = 1.1, uplimit = 300, crossPart = "EQU")
```

```
## with 3 crossover parts and equal locations
crossover(CrossSampl, u = 2.5, uplimit = 300, crossPart = "EQU")

## or with random locations and 5 crossover parts
crossover(CrossSampl, u = 4.9, uplimit = 300, crossPart = "RAN")
```

`fitness` *Evaluate the Individual Fitness values*

Description

The fitness of all individuals in the current population is calculated after their energy output has been evaluated in `calculate_energy`. This function reduces the resulting energy outputs to a single fitness value for each individual.

Usage

```
fitness(
  selection,
  referenceHeight,
  RotorHeight,
  SurfaceRoughness,
  Polygon,
  resol1,
  rot,
  dirspeed,
  srtm_crop,
  topograp,
  cclRaster,
  weibull,
  Parallel,
  numCluster
)
```

Arguments

`selection`	A list containing all individuals of the current population.
`referenceHeight`	The height at which the incoming wind speeds were measured. Default is `RotorHeight`
`RotorHeight`	The height of the turbine hub
`SurfaceRoughness`	A surface roughness length in meters. With the terrain effect model, a surface roughness is calculated for every grid cell using the elevation and land cover data. Default is `0.3`

Polygon	The considered area as shapefile.
resol1	The resolution of the grid in meter.
rot	The desired rotor radius in meter.
dirspeed	The wind data as list.
srtm_crop	A list of 3 raster, with 1) the elevation, 2) an orographic and 3) a terrain raster. Calculated in genetic_algorithm
topograp	Boolean value, which indicates if the terrain effect model should be enabled or not. Default is FALSE
cclRaster	A Corine Land Cover raster, that has to be adapted previously by hand with the surface roughness length for every land cover type. Is only used, when the terrain effect model is activated.
weibull	A raster representing the estimated wind speeds
Parallel	A boolean value, indicating whether parallel processing should be used. The *parallel* and *doParallel* packages are used for parallel processing. Default is FALSE
numCluster	If Parallel is TRUE, this variable defines the number of clusters to be used. Default is 2

Value

Returns a list with every individual, consisting of X & Y coordinates, rotor radii, the runs and the selected grid cell IDs, and the resulting energy outputs, efficiency rates and fitness values.

See Also

Other Genetic Algorithm Functions: crossover(), genetic_algorithm(), init_population(), mutation(), selection(), trimton()

Examples

```
## Create a random rectangular shapefile
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

## Create a uniform and unidirectional wind data.frame and plots the
## resulting wind rose
## Uniform wind speed and single wind direction
wind <- data.frame(ws = 12, wd = 0)
# windrosePlot <- plot_windrose(data = wind, spd = wind$ws,
#                dir = wind$wd, dirres=10, spdmax=20)

## Calculate a Grid and an indexed data.frame with coordinates and
```

```
## grid cell IDs.
Grid1 <- grid_area(shape = Polygon1, size = 200, prop = 1)
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

wind <- list(wind, probab = 100)
startsel <- init_population(Grid, 10, 20)
fit <- fitness(
  selection = startsel, referenceHeight = 100, RotorHeight = 100,
  SurfaceRoughness = 0.3, Polygon = Polygon1, resol1 = 200, rot = 20,
  dirspeed = wind, srtm_crop = "", topograp = FALSE, cclRaster = "",
  Parallel = FALSE
)
```

---

genetic_algorithm *Run a Genetic Algorithm to optimize a wind farm layout*

Description

Run a Genetic Algorithm to optimize the layout of wind turbines on a given area. The algorithm works with a fixed amount of turbines, a fixed rotor radius and a mean wind speed value for every incoming wind direction.

Usage

```
genetic_algorithm(
  Polygon1,
  GridMethod,
  Rotor,
  n,
  fcrR,
  referenceHeight,
  RotorHeight,
  SurfaceRoughness,
  Proportionality,
  iteration,
  mutr,
  vdirspe,
  topograp,
  elitism,
  nelit,
  selstate,
  crossPart1,
  trimForce,
  Projection,
  sourceCCL,
  sourceCCLRoughness,
```

```
  weibull,
  weibullsrc,
  Parallel,
  numCluster,
  verbose = FALSE,
  plotit = FALSE
)
```

Arguments

Polygon1	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
GridMethod	Should the polygon be divided into rectangular or hexagonal grid cells? The default is `Rectangular` grid. Hexagonal grids are computed when assigning `h` or `hexagon` to this input variable.
Rotor	The rotor radius in meter
n	The amount of turbines
fcrR	A numeric value used for grid spacing. Default is `5`
referenceHeight	The height at which the incoming wind speeds were measured. Default is `RotorHeight`
RotorHeight	The height of the turbine hub
SurfaceRoughness	A surface roughness length in meters. With the terrain effect model, a surface roughness is calculated for every grid cell using the elevation and land cover data. Default is `0.3`
Proportionality	A numeric value used for the grid calculation, as it determines the percentage a grid cell must overlay the area. Default is `1`
iteration	The number of iterations. Default is `20`
mutr	A numeric mutation rate. Default is `0.008`
vdirspe	A data.frame containing the wind speeds, directions and probabilities. See windata_format.
topograp	Boolean value, which indicates if the terrain effect model should be enabled or not. Default is `FALSE`
elitism	Boolean value, which indicates whether elitism should be activated or not. Default is `TRUE`
nelit	If `elitism` is TRUE, this input determines the amount of individuals in the elite group. Default is 7
selstate	Determines which selection method is used, "FIX" selects a constant percentage and "VAR" selects a variable percentage, depending on the development of the fitness values. Default is "FIX"
crossPart1	Determines which crossover method is used, "EQU" divides the genetic code at equal intervals and "RAN" divides the genetic code at random locations. Default is `"EQU"`

trimForce	If TRUE the algorithm will use a probabilistic approach to correct the windfarms to the desired amount of turbines. If FALSE the adjustment will be random. Default is FALSE
Projection	A spatial reference system. Depending on your PROJ-version, it should either be a numeric EPSG-code or a Proj4-string. Default is EPSG:3035
sourceCCL	The path to the Corine Land Cover raster (.tif). Only required when the terrain effect model is activated.
sourceCCLRoughness	The source to the adapted Corine Land Cover legend as .csv file. Only required when terrain effect model is activated. As default a .csv file within this package ('~/extdata') is taken that was already adapted manually.
weibull	A boolean value that specifies whether to take Weibull parameters into account. If TRUE, the wind speed values of vdirspe are ignored. The algorithm will calculate the mean wind speed for every wind turbine according to the Weibull parameters. Default is FALSE
weibullsrc	A list of Weibull parameter rasters, where the first list item must be the shape parameter raster k and the second item must be the scale parameter raster a of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria. This variable is only used if weibull = TRUE.
Parallel	A boolean value, indicating whether parallel processing should be used. The *parallel* and *doParallel* packages are used for parallel processing. Default is FALSE
numCluster	If Parallel is TRUE, this variable defines the number of clusters to be used. Default is 2
verbose	If TRUE it will print information for every generation. Default is FALSE
plotit	If TRUE it will plot the best windfarm of every generation. Default is FALSE

Details

A terrain effect model can be included in the optimization process. Therefore, a digital elevation model will be downloaded automatically via the elevatr::get_elev_raster function. A land cover raster can also downloaded automatically from the EEA-website, or the path to a raster file can be passed to sourceCCL. The algorithm uses an adapted version of the Raster legend ("clc_legend.csv"), which is stored in the package directory '~/inst/extdata'. To use other values for the land cover roughness lengths, insert a column named **"Rauhigkeit_z"** to the .csv file, assign a surface roughness length to all land cover types. Be sure that all rows are filled with numeric values and save the file with **";"** separation. Assign the path of the file to the input variable sourceCCLRoughness of this function.

Value

The result is a matrix with aggregated values per generation; the best individual regarding energy and efficiency per generation, some fuzzy control variables per generation, a list of all fitness values per generation, the amount of individuals after each process, a matrix of all energy, efficiency and fitness values per generation, the selection and crossover parameters, a matrix with the generational difference in maximum and mean energy output, a matrix with the given inputs, a dataframe with the wind information, the mutation rate per generation and a matrix with all tested wind farm layouts.

See Also

Other Genetic Algorithm Functions: crossover(), fitness(), init_population(), mutation(), selection(), trimton()

Examples

```
## Not run:
## Create a random rectangular shapefile
library(sf)

Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plot_windrose(
  data = data.in, spd = data.in$ws,
  dir = data.in$wd, dirres = 10, spdmax = 20
)

## Runs an optimization run for 20 iterations with the
## given shapefile (Polygon1), the wind data.frame (data.in),
## 12 turbines (n) with rotor radii of 30m (Rotor) and rotor height of 100m.
result <- genetic_algorithm(
  Polygon1 = Polygon1,
  n = 12,
  vdirspe = data.in,
  Rotor = 30,
  RotorHeight = 100
)
plot_windfarmGA(result = result, Polygon1 = Polygon1)

## End(Not run)
```

---

get_dist_angles *Calculate distances and angles of possibly influencing turbines*

Description

Calculate distances and angles for a turbine and all it's potentially influencing turbines.

Usage

```
get_dist_angles(t, o, wnkl, dist, polYgon, plotAngles = FALSE)
```

Arguments

t	A data.frame of the current individual with X and Y coordinates
o	A numeric value indicating the index of the current turbine
wnkl	The angle from which wake influences are considered to be negligible
dist	A numeric value indicating the distance, after which the wake effects are considered to be eliminated.
polYgon	A shapefile representing the considered area
plotAngles	A logical variable, which is used to plot the distances and angles. Default is FALSE

Value

Returns a matrix with the distances, angles and heights of potentially influencing turbines

See Also

Other Wind Energy Calculation Functions: barometric_height(), calculate_energy(), circle_intersection(), turbine_influences()

Examples

```
library(sf)

## Exemplary input Polygon with 2km x 2km:
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

## Create a random windfarm with 10 turbines
t <- st_coordinates(st_sample(Polygon1, 10))
t <- cbind(t, "Z" = 1)
wnkl <- 20
dist <- 100000

## Evaluate and plot for every turbine all other potentially influencing turbines
potInfTur <- list()
for (i in 1:(length(t[, 1]))) {
  potInfTur[[i]] <- get_dist_angles(
    t = t, o = i, wnkl = wnkl,
    dist = dist, polYgon = Polygon1, plotAngles = TRUE
  )
}
potInfTur
```

get_grids *Get the Grid-IDs from binary matrix*

Description

Retrieve the grid ID's from the binary matrix, where the binary code indicates which grid cells are used in the current wind farm constellation.

Usage

```
get_grids(trimtonOut, Grid)
```

Arguments

trimtonOut	Input matrix with binary values
Grid	Grid of the considered area

Value

Returns a list of all individuals with X and Y coordinates and the grid cell ID.

See Also

Other Helper Functions: grid_area(), hexa_area(), isSpatial(), permutations(), readinteger(), readintegerSel(), splitAt(), windata_format()

Examples

```
## Create a random rectangular shapefile
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(0, 0, 2000, 2000, 0),
    c(0, 2000, 2000, 0, 0)
  ))),
  crs = 3035
))

## Calculate a Grid and an indexed data.frame with coordinates and
## grid cell Ids.
Grid1 <- grid_area(shape = Polygon1, size = 200, prop = 1)
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- init_population(Grid, 10, 20)
wind <- data.frame(ws = 12, wd = 0)
wind <- list(wind, probab = 100)
fit <- fitness(
  selection = startsel, referenceHeight = 100, RotorHeight = 100,
```

```
  SurfaceRoughness = 0.3, Polygon = Polygon1, resol1 = 200, rot = 20,
  dirspeed = wind, srtm_crop = "", topograp = FALSE, cclRaster = ""
)
allparks <- do.call("rbind", fit)

## SELECTION
## print the amount of Individuals selected.
## Check if the amount of Turbines is as requested.
selec6best <- selection(fit, Grid, 2, TRUE, 6, "VAR")

## CROSSOVER
## u determines the amount of crossover points,
## crossPart determines the method used (Equal/Random),
## uplimit is the maximum allowed permutations
crossOut <- crossover(selec6best, 2, uplimit = 300, crossPart = "RAN")

## MUTATION
## Variable Mutation Rate is activated if more than 2 individuals represent
## the current best solution.
mut <- mutation(a = crossOut, p = 0.3)

## TRIMTON
## After Crossover and Mutation, the amount of turbines in a windpark change
## and have to be corrected to the required amount of turbines.
mut1 <- trimton(
  mut = mut, nturb = 10, allparks = allparks,
  nGrids = AmountGrids, trimForce = FALSE
)

## Get the new Grid-Ids and run a new fitness run.
getRectV <- get_grids(mut1, Grid)
fit <- fitness(
  selection = getRectV, referenceHeight = 100, RotorHeight = 100,
  SurfaceRoughness = 0.3, Polygon = Polygon1, resol1 = 200, rot = 20,
  dirspeed = wind, srtm_crop = "", topograp = FALSE, cclRaster = ""
)
head(fit)
```

`grid_area` *Make a grid from a Simple Feature Polygon*

Description

Create a grid from a given polygon with a certain resolution and proportionality. The grid cell centroids represent possible wind turbine locations.

Usage

```
grid_area(shape, size = 500, prop = 1, plotGrid = FALSE)
```

Arguments

shape	Simple Feature Polygon of the considered area
size	The cellsize of the grid in meters. Default is 500
prop	A factor used for grid calculation. It determines the minimum percentage that a grid cell must cover the area. Default is 1
plotGrid	Logical value indicating whether the results should be plotted. Default is FALSE

Value

Returns a list with 2 elements. List element 1 will have the grid cell IDS, and the X and Y coordinates of the centers of each grid cell. List element 2 is the grid as Simple Feature Polygons, which is used for plotting purposes.

Note

The grid of the genetic algorithm will have a resolution of Rotor * fcrR. See the arguments of genetic_algorithm

See Also

Other Helper Functions: get_grids(), hexa_area(), isSpatial(), permutations(), readinteger(), readintegerSel(), splitAt(), windata_format()

Examples

```
## Exemplary input Polygon with 2km x 2km:
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(0, 0, 2000, 2000, 0),
    c(0, 2000, 2000, 0, 0)
  ))),
  crs = 3035
))

## Create a Grid
grid_area(Polygon1, 200, 1, TRUE)
grid_area(Polygon1, 400, 1, TRUE)

## Examplary irregular input Polygon
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(0, 0, 2000, 3000, 0),
    c(20, 200, 2000, 0, 20)
  ))),
  crs = 3035
))

## Create a Grid
grid_area(Polygon1, 200, 1, TRUE)
```

```
grid_area(Polygon1, 200, 0.1, TRUE)
grid_area(Polygon1, 400, 1, TRUE)
grid_area(Polygon1, 400, 0.1, TRUE)
```

---

hexa_area *Polygon to Hexagonal Grids*

Description

The function takes a Simple Feature Polygon and a size argument and creates a list with an indexed matrix with coordinates and a Simple Feature object, that consists of hexagonal grids.

Usage

```
hexa_area(shape, size = 500, plotGrid = FALSE)
```

Arguments

shape	Simple Feature Polygon of the considered area
size	The cellsize of the grid in meters. Default is 500
plotGrid	Logical value indicating whether the results should be plotted. Default is FALSE

Value

Returns a list with 2 elements. List element 1 will have the grid cell IDS, and the X and Y coordinates of the centers of each grid cell. List element 2 is the grid as Simple Feature Polygons, which is used for plotting purposes.

See Also

Other Helper Functions: get_grids(), grid_area(), isSpatial(), permutations(), readinteger(), readintegerSel(), splitAt(), windata_format()

Examples

```
library(sf)
## Exemplary input Polygon with 2km x 2km:
Poly <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))
HexGrid <- hexa_area(Poly, 100, TRUE)
```

hole_shape *A POLYGON with a hole*

Description

A POLYGON with a hole

Usage

```
hole_shape
```

Format

An object of class `sf` (inherits from `data.frame`) with 1 rows and 2 columns.

init_population *Create a random initial Population*

Description

Create `nStart` random sub-selections from the indexed grid and assign binary variable 1 to selected grids. This function initiates the genetic algorithm with a first random population and will only be needed in the first iteration.

Usage

```
init_population(Grid, n, nStart = 100)
```

Arguments

`Grid`	The data.frame output of `grid_area`" function, with X and Y coordinates and Grid cell IDs.
`n`	A numeric value indicating the amount of required turbines.
`nStart`	A numeric indicating the amount of randomly generated initial individuals. Default is 100.

Value

Returns a list of `nStart` initial individuals, each consisting of `n` turbines. Resulting list has the x and y coordinates, the grid cell ID and a binary variable of 1, indicating a turbine in the grid cell.

See Also

Other Genetic Algorithm Functions: `crossover()`, `fitness()`, `genetic_algorithm()`, `mutation()`, `selection()`, `trimton()`

Examples

```
library(sf)
## Exemplary input Polygon with 2km x 2km:
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

Grid <- grid_area(Polygon1, 200, 1, TRUE)

## Create 5 individuals with 10 wind turbines each.
firstPop <- init_population(Grid = Grid[[1]], n = 10, nStart = 5)
```

`isSpatial` *Transform to Simple Feature Polygons*

Description

Helper Function, which transforms SpatialPolygons or coordinates in matrix/data.frame - form to a Simple Feature Polygon

Usage

```
isSpatial(shape, proj)
```

Arguments

`shape`	An area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
`proj`	Which Projection should be assigned to matrix / data.frame coordinates

Details

If the columns are named, it will look for common abbreviation to match x/y or long/lat columns. If the columns are not named, the first 2 numeric columns are taken.

Value

A Simple Feature Polygon

See Also

Other Helper Functions: `get_grids()`, `grid_area()`, `hexa_area()`, `permutations()`, `readinteger()`, `readintegerSel()`, `splitAt()`, `windata_format()`

Examples

```
library(sf)
df <- rbind(
  c(4498482, 2668272), c(4498482, 2669343),
  c(4499991, 2669343), c(4499991, 2668272)
)
isSpatial(df)

Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))
isSpatial(st_coordinates(Polygon1), 3035)
```

---

multi_shape *A MULTIPOLYGON with 3 Polygons*

Description

A MULTIPOLYGON with 3 Polygons

Usage

```
multi_shape
```

Format

An object of class `sf` (inherits from `data.frame`) with 1 rows and 1 columns.

---

mutation *Mutation Method*

Description

The function randomly mutates an individual's genetic code

Usage

```
mutation(a, p, seed = NULL)
```

Arguments

a	The binary matrix of all individuals
p	The mutation rate
seed	Set a seed for comparability. Default is NULL

Value

Returns a binary matrix with mutated genes.

See Also

Other Genetic Algorithm Functions: crossover(), fitness(), genetic_algorithm(), init_population(), selection(), trimton()

Examples

```
## Create 4 random individuals with binary values
a <- cbind(
  bin0 = sample(c(0, 1), 20, replace = TRUE, prob = c(70, 30)),
  bin1 = sample(c(0, 1), 20, replace = TRUE, prob = c(30, 70)),
  bin2 = sample(c(0, 1), 20, replace = TRUE, prob = c(30, 70)),
  bin3 = sample(c(0, 1), 20, replace = TRUE, prob = c(30, 70))
)
a

## Mutate the individuals with a low percentage
aMut <- mutation(a, 0.1, NULL)
## Check which values are not like the originals
a == aMut

## Mutate the individuals with a high percentage
aMut <- mutation(a, 0.4, NULL)
## Check which values are not like the originals
a == aMut
```

---

package_installed *Is the package installed or not*

Description

Is the package installed or not

Usage

```
is_foreach_installed()

is_parallel_installed()

is_doparallel_installed()

is_ggplot2_installed()

is_leaflet_installed()

is_elevatr_installed()
```

Value

An invisible boolean value, indicating if the package is installed or not.

---

permutations *Enumerate the Combinations or Permutations of the Elements of a Vector*

Description

permutations enumerates the possible permutations. The function is forked and minified from gtools::permutations

Usage

```
permutations(n, r, v = 1:n)
```

Arguments

n	Size of the source vector
r	Size of the target vectors
v	Source vector. Defaults to 1:n

Value

Returns a matrix where each row contains a vector of length r.

Author(s)

Original versions by Bill Venables. Extended to handle repeats.allowed by Gregory R. Warnes

References

Venables, Bill. "Programmers Note", R-News, Vol 1/1, Jan. 2001. https://cran.r-project.org/doc/Rnews/

See Also

Other Helper Functions: get_grids(), grid_area(), hexa_area(), isSpatial(), readinteger(), readintegerSel(), splitAt(), windata_format()

---

plot_cloud *Plot outputs of all generations with standard deviations*

Description

Plot the fitness, efficiency and energy outputs of all generations and the corresponding standard deviations.

Usage

```
plot_cloud(result, pl = FALSE)
```

Arguments

result	The output of genetic_algorithm
pl	Should the results be plotted? Default is FALSE

Value

Returns a data.frame with the values for fitness, efficiency and energy for all evaluated individuals

See Also

Other Plotting Functions: plot_development(), plot_evolution(), plot_fitness_evolution(), plot_parkfitness(), plot_result(), plot_windfarmGA(), plot_windrose(), random_search_single()

Examples

```
## Plot the results of a hexagonal grid optimization
plcdf <- plot_cloud(resulthex, TRUE)
```

plot_development *Plot the progress of populations*

Description

Plot the changes in mean and max fitness values to previous generation.

Usage

```
plot_development(result)
```

Arguments

result	The output of genetic_algorithm

Value

Returns NULL. Used for plotting

See Also

Other Plotting Functions: plot_cloud(), plot_evolution(), plot_fitness_evolution(), plot_parkfitness(), plot_result(), plot_windfarmGA(), plot_windrose(), random_search_single()

Examples

```
plot_development(resultrect)
```

plot_evolution *Plot the evolution of fitness values*

Description

Plot the evolution of energy outputs and efficiency rates over the whole generations. Plots min, mean and max values.

Usage

```
plot_evolution(result, ask = TRUE, spar = 0.1)
```

Arguments

result	The output of genetic_algorithm
ask	Should R wait for interaction for subsequent plotting. Default is TRUE
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1

Value

Returns NULL. Used for plotting

See Also

Other Plotting Functions: plot_cloud(), plot_development(), plot_fitness_evolution(), plot_parkfitness(), plot_result(), plot_windfarmGA(), plot_windrose(), random_search_single()

Examples

```
## Plot the results of a rectangular grid optimization
plot_evolution(resultrect, ask = TRUE, spar = 0.1)
```

---

plot_fitness_evolution

Plot the changes of min/mean/max fitness values

Description

Plot the evolution of fitness values and the change in the min, mean and max fitness values to the former generations.

Usage

```
plot_fitness_evolution(result, spar = 0.1)
```

Arguments

result	The output of genetic_algorithm
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1

Value

Returns NULL. Used for plotting

See Also

Other Plotting Functions: plot_cloud(), plot_development(), plot_evolution(), plot_parkfitness(), plot_result(), plot_windfarmGA(), plot_windrose(), random_search_single()

Examples

```
## Plot the results of a hexagonal grid optimization
plot_fitness_evolution(resulthex, 0.1)
```

plot_leaflet *Plot a wind warm with leaflet*

Description

Plot a resulting wind farm on a leaflet map.

Usage

```
plot_leaflet(result, Polygon1, which = 1, orderitems = TRUE, GridPol)
```

Arguments

result	The output of genetic_algorithm
Polygon1	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
which	A numeric value, indicating which individual to plot. The default is 1. Combined with orderitems = TRUE this will show the best performing wind farm.
orderitems	A logical value indicating whether the results should be ordered by energy values TRUE or chronologically FALSE
GridPol	By default, the grid will be calculated based on the inputs of result and the Polygon1. But another spatial object or the output of the grid_area or hexa_area functions can also be

Value

Returns a leaflet map.

Examples

```
## Not run:
## Plot the best wind farm on a leaflet map (ordered by energy values)
plot_leaflet(result = resulthex, Polygon1 = sp_polygon, which = 1)

## Plot the last wind farm (ordered by chronology).
plot_leaflet(
  result = resulthex, Polygon1 = sp_polygon, orderitems = FALSE,
  which = 1
)

## Plot the best wind farm on a leaflet map with the rectangular Grid
Grid <- grid_area(sp_polygon, size = 150, prop = 0.4)
plot_leaflet(
  result = resultrect, Polygon1 = sp_polygon, which = 1,
  GridPol = Grid[[2]]
)

## Plot the last wind farm with hexagonal Grid
```

```
Grid <- hexa_area(sp_polygon, size = 75)
plot_leaflet(
  result = resulthex, Polygon1 = sp_polygon, which = 1,
  GridPol = Grid[[2]]
)

## End(Not run)
```

`plot_parkfitness` *Plot the genetic algorithm results*

Description

Plot the evolution of fitness values with the influences of selection, crossover and mutation.

Usage

```
plot_parkfitness(result, spar = 0.1)
```

Arguments

`result`	The output of `genetic_algorithm`
`spar`	A numeric value determining how exact a spline should be drawn. Default is 0.1

Value

Returns NULL. Used for plotting

See Also

Other Plotting Functions: `plot_cloud()`, `plot_development()`, `plot_evolution()`, `plot_fitness_evolution()`, `plot_result()`, `plot_windfarmGA()`, `plot_windrose()`, `random_search_single()`

Examples

```
## Plot the results of a hexagonal grid optimization
plot_parkfitness(resulthex)
```

plot_random_search *Plot the result of a randomized output.*

Description

Plotting method for the results of random_search_single and random_search.

Usage

```
plot_random_search(resultRS, result, Polygon1, best)
```

Arguments

resultRS	The result of the random functions random_search_single and random_search.
result	The output of genetic_algorithm
Polygon1	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
best	How many best candidates to plot. Default is 1.

Value

Returns NULL. Used for plotting

See Also

Other Randomization: random_search(), random_search_single()

Examples

```
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

Res <- random_search(result = resultrect, Polygon1 = Polygon1)
plot_random_search(resultRS = Res, result = resultrect, Polygon1 = Polygon1, best = 2)
```

plot_result *Plot the best results*

Description

Plot the best solutions of the genetic algorithm. Depending on `plotEn`, either the best energy or efficiency solutions can be plotted. `best` indicates the amount of best solutions to plot.

Usage

```
plot_result(
  result,
  Polygon1,
  best = 3,
  plotEn = 1,
  topographie = FALSE,
  Grid = TRUE,
  sourceCCLRoughness = NULL,
  sourceCCL = NULL,
  weibullsrc
)
```

Arguments

`result`	The output of `genetic_algorithm`
`Polygon1`	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
`best`	A numeric value indicating how many of the best individuals should be plotted
`plotEn`	A numeric value that indicates if the best energy or efficiency output should be plotted. `1` plots the best energy solutions and `2` plots the best efficiency solutions
`topographie`	A logical value, indicating whether terrain effects should be considered and plotted or not
`Grid`	If `TRUE` (default) the used grid will be added to the plot. You can also pass another Simple Feature object
`sourceCCLRoughness`	The source to the adapted Corine Land Cover legend as .csv file. Only required when terrain effect model is activated. As default a .csv file within this package ('`~/extdata`') is taken that was already adapted manually.
`sourceCCL`	The path to the Corine Land Cover raster (.tif). Only required when the terrain effect model is activated.
`weibullsrc`	A list of Weibull parameter rasters, where the first list item must be the shape parameter raster `k` and the second item must be the scale parameter raster `a` of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria. This variable is only used if `weibull = TRUE`.

Value

Returns a data.frame of the best (energy/efficiency) individual during all iterations

See Also

Other Plotting Functions: plot_cloud(), plot_development(), plot_evolution(), plot_fitness_evolution(), plot_parkfitness(), plot_windfarmGA(), plot_windrose(), random_search_single()

Examples

```
## Not run:
## Add some data examples from the package
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

## Plot the results of a hexagonal grid optimization
plot_result(resulthex, Polygon1, best = 1, plotEn = 1, topographie = FALSE)

## Plot the results of a rectangular grid optimization
plot_result(resultrect, Polygon1, best = 1, plotEn = 1, topographie = FALSE)

## End(Not run)
```

---

plot_viewshed *Plot visibility*

Description

Calculate and plot visibility for given points in a given area.

Usage

```
plot_viewshed(r, turbine_locs, h1 = 0, h2 = 0, plot = TRUE, ...)
```

Arguments

r	The elevation SpatRaster
turbine_locs	Coordinates, SpatialPoint or SimpleFeature Points representing the wind turbines
h1	A single number or numeric vector giving the extra height offsets for the turbine_locs
h2	The height offset for Point 2
plot	Should the result be plotted. Default is TRUE
...	forwarded to terra::plot

Value

A mosaiced SpatRaster, representing the visibility for all `turbine_locs`

Examples

```
library(sf)
library(terra)

f <- system.file("ex/elev.tif", package = "terra")
r <- rast(f)
x <- project(r, "EPSG:2169")
shape <- sf::st_as_sf(as.polygons(terra::boundaries(x)))
plot(shape)
st_crs(shape) <- 2169
locs <- st_sample(shape, 10, type = "random")
plot_viewshed(x, locs, h1 = 0, h2 = 0, plot = TRUE)
```

---

plot_windfarmGA *Plot the results of an optimization run*

Description

Plot the results of a genetic algorithm run with given inputs. Several plots try to show all relevant effects and outcomes of the algorithm. 6 plot methods are available that can be selected individually.

Usage

```
plot_windfarmGA(
  result,
  Polygon1,
  whichPl = "all",
  best = 1,
  plotEn = 1,
  weibullsrc
)
```

Arguments

`result`	The output of `genetic_algorithm`
`Polygon1`	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
`whichPl`	Which plots should be shown: 1-6 are possible. The default is "all" which shows all available plots
`best`	A numeric value indicating how many of the best individuals should be plotted
`plotEn`	A numeric value that indicates if the best energy or efficiency output should be plotted. 1 plots the best energy solutions and 2 plots the best efficiency solutions

`weibullsrc` A list of Weibull parameter rasters, where the first list item must be the shape parameter raster `k` and the second item must be the scale parameter raster `a` of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria. This variable is only used if `weibull = TRUE`.

Value

Returns NULL. Used for plotting

See Also

Other Plotting Functions: `plot_cloud()`, `plot_development()`, `plot_evolution()`, `plot_fitness_evolution()`, `plot_parkfitness()`, `plot_result()`, `plot_windrose()`, `random_search_single()`

Examples

```
## Not run:
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

## Plot the results of a hexagonal grid optimization
plot_windfarmGA(resulthex, Polygon1, whichPl = "all", best = 1, plotEn = 1)

## Plot the results of a rectangular grid optimization
plot_windfarmGA(resultrect, Polygon1, whichPl = "all", best = 1, plotEn = 1)

## End(Not run)
```

---

plot_windrose *Plot a Windrose*

Description

Plot a wind rose of the wind data frame.

Usage

```
plot_windrose(
  data,
  spd,
  dir,
  spdres = 2,
```

```
  dirres = 10,
  spdmin = 1,
  spdmax = 30,
  palette = "YlGnBu",
  spdseq = NULL,
  plotit = TRUE
)
```

Arguments

data	A data.frame containing the wind information
spd	The column of the wind speeds in "data"
dir	The column of the wind directions in "data"
spdres	The increment of the wind speed legend. Default is 2
dirres	The size of the wind sectors. Default is 10
spdmin	Minimum wind speed. Default is 1
spdmax	Maximal wind speed. Default is 30
palette	A color palette used for drawing the wind rose
spdseq	A wind speed sequence, that is used for plotting
plotit	Should the windrose be plotted? Default is TRUE

Value

Returns NULL. Used for plotting

See Also

Other Plotting Functions: plot_cloud(), plot_development(), plot_evolution(), plot_fitness_evolution(), plot_parkfitness(), plot_result(), plot_windfarmGA(), random_search_single()

Examples

```
## Exemplary Input Wind speed and direction data frame
# Uniform wind speed and single wind direction
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plot_windrose(
  data = data.in, spd = data.in$ws,
  dir = data.in$wd
)

# Random wind speeds and random wind directions
data.in <- data.frame(
  ws = sample(1:25, 10),
  wd = sample(1:260, 10)
)
windrosePlot <- plot_windrose(
  data = data.in, spd = data.in$ws,
  dir = data.in$wd
```

```
)
```

`random_search` *Randomize the output of the Genetic Algorithm*

Description

Perform a random search in the grid cells, to further optimize the output of the wind farm layout.

Usage

```
random_search(result, Polygon1, n = 20, best = 1, Plot = FALSE, max_dist = 2.2)
```

Arguments

`result`	The resulting matrix of the function `genetic_algorithm`
`Polygon1`	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
`n`	The number of random searches to be performed. Default is 20.
`best`	Which best individuals should be the starting conditions for a random search. The default is 1.
`Plot`	Should the random search be plotted? Default is `FALSE`
`max_dist`	A numeric value multiplied by the rotor radius to perform collision checks. Default is `2.2`

Value

Returns a list.

See Also

Other Randomization: `plot_random_search()`, `random_search_single()`

Examples

```
new <- random_search(resultrect, sp_polygon, n = 20, best = 4)
plot_random_search(resultRS = new, result = resultrect, Polygon1 = sp_polygon, best = 2)
```

random_search_single *Randomize the location of a single turbine*

Description

Perform a random search for a single turbine, to further optimize the output of the wind farm layout.

Usage

```
random_search_single(result, Polygon1, n = 20, Plot = FALSE, max_dist = 2.2)
```

Arguments

result	The resulting matrix of the function genetic_algorithm
Polygon1	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
n	The number of random searches to be performed. Default is 20.
Plot	Should the random search be plotted? Default is FALSE
max_dist	A numeric value multiplied by the rotor radius to perform collision checks. Default is 2.2

Value

Returns a list

See Also

Other Randomization: plot_random_search(), random_search()

Other Plotting Functions: plot_cloud(), plot_development(), plot_evolution(), plot_fitness_evolution(), plot_parkfitness(), plot_result(), plot_windfarmGA(), plot_windrose()

readinteger *Check Input Crossover Method*

Description

Checks whether the input for crossover is given correctly. If not, a message is prompted which asks to input one of the 2 available crossover methods. The available inputs are "E" and "R". "E" refers to partitioning at equal intervals and "R" refers to random partitioning.

Usage

```
readinteger()
```

Value

Returns the selected crossover method (character)

See Also

Other Helper Functions: get_grids(), grid_area(), hexa_area(), isSpatial(), permutations(), readintegerSel(), splitAt(), windata_format()

---

readintegerSel *Check Input Selection Method*

Description

Checks whether the input for selection is given correctly. If not, a message is prompted which asks to input one of the 2 available selection methods. The available inputs are "F" and "V". "F" refers to a fixed percentage of 50% and "V" refers to a variable percentage, based on the development of the population fitness values.

Usage

```
readintegerSel()
```

Value

Returns the selected selection method (character)

See Also

Other Helper Functions: get_grids(), grid_area(), hexa_area(), isSpatial(), permutations(), readinteger(), splitAt(), windata_format()

---

resulthex *A resulting matrix of* genetic_algorithm *with 10 iterations and a hexagonal grid derived from* sp_polygon

---

Description

A resulting matrix of genetic_algorithm with 10 iterations and a hexagonal grid derived from sp_polygon

Usage

```
resulthex
```

Format

An object of class matrix (inherits from array) with 10 rows and 13 columns.

resultrect — *A resulting matrix of* genetic_algorithm *with 200 iterations and a rectangular grid derived from* sp_polygon

Description

A resulting matrix of genetic_algorithm with 200 iterations and a rectangular grid derived from sp_polygon

Usage

```
resultrect
```

Format

An object of class matrix (inherits from array) with 200 rows and 13 columns.

selection — *Selection Method*

Description

Select a certain amount of individuals and recombine them to parental teams. Add the mean fitness value of both parents to the parental team. Depending on the selected selstate, the algorithm will either take always 50 percent or a variable percentage of the current population. The variable percentage depends on the evolution of the populations fitness values.

Usage

```
selection(fit, Grid, teil, elitism, nelit, selstate, verbose)
```

Arguments

fit	A list of all fitness-evaluated individuals
Grid	Is the indexed grid output from grid_area
teil	A numeric value that determines the selection percentage
elitism	Boolean value, which indicates whether elitism should be activated or not. Default is TRUE
nelit	If elitism is TRUE, this input determines the amount of individuals in the elite group. Default is 7
selstate	Determines which selection method is used, "FIX" selects a constant percentage and "VAR" selects a variable percentage, depending on the development of the fitness values. Default is "FIX"
verbose	If TRUE, will print out further information.

Value

Returns list with 2 elements. Element 1 is the binary encoded matrix which shows all selected individuals. Element 2 represent the mean fitness values of each parental team.

See Also

Other Genetic Algorithm Functions: crossover(), fitness(), genetic_algorithm(), init_population(), mutation(), trimton()

Examples

```
## Exemplary input Polygon with 2km x 2km:
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4498482, 4498482, 4499991, 4499991, 4498482),
    c(2668272, 2669343, 2669343, 2668272, 2668272)
  ))),
  crs = 3035
))

## Calculate a Grid and an indexed data.frame with coordinates and grid cell Ids.
Grid1 <- grid_area(shape = Polygon1, size = 200, prop = 1)
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- init_population(Grid, 10, 20)
wind <- as.data.frame(cbind(ws = 12, wd = 0))
wind <- list(wind, probab = 100)
fit <- fitness(
  selection = startsel, referenceHeight = 100, RotorHeight = 100,
  SurfaceRoughness = 0.3, Polygon = Polygon1, resol1 = 200,
  rot = 20, dirspeed = wind,
  srtm_crop = "", topograp = FALSE, cclRaster = ""
)
allparks <- do.call("rbind", fit)
## SELECTION
## print the amount of Individuals selected. Check if the amount
## of Turbines is as requested.
selec6best <- selection(fit, Grid, 2, TRUE, 6, "VAR")
selec6best <- selection(fit, Grid, 2, TRUE, 6, "FIX")
selec6best <- selection(fit, Grid, 4, FALSE, 6, "FIX")
```

---

splitAt *Split matrices or numeric vectors at specific indices*

Description

The function is used by the crossover method to split a genetic code at certain intervals. See also crossover.

Usage

```
splitAt(x, pos)
```

Arguments

x	A numeric variable that represents an individual's binary genetic code
pos	A numeric value that indicates where to split the genetic code

Value

Returns a list of the split genetic code.

See Also

Other Helper Functions: get_grids(), grid_area(), hexa_area(), isSpatial(), permutations(), readinteger(), readintegerSel(), windata_format()

Examples

```
splitAt(1:100, 20)
splitAt(as.matrix(1:100), 20)
```

---

sp_polygon *The rectangular POLYGON used to create* resultrect *&* resulthex

---

Description

The rectangular POLYGON used to create resultrect & resulthex

Usage

```
sp_polygon
```

Format

An object of class sf (inherits from data.frame) with 1 rows and 1 columns.

`terrain_model` *Get topographic rasters*

Description

Calculate the SpatRasters needed for the terrain model.

Usage

```
terrain_model(
  topograp = TRUE,
  Polygon1,
  sourceCCL,
  sourceCCLRoughness,
  plotit = FALSE,
  verbose = FALSE
)
```

Arguments

`topograp`	Boolean value, which indicates if the terrain effect model should be enabled or not. Default is `FALSE`
`Polygon1`	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
`sourceCCL`	The path to the Corine Land Cover raster (.tif). Only required when the terrain effect model is activated.
`sourceCCLRoughness`	The source to the adapted Corine Land Cover legend as .csv file. Only required when terrain effect model is activated. As default a .csv file within this package ('`~/extdata`') is taken that was already adapted manually.
`plotit`	Plots the elevation data
`verbose`	If TRUE it will print information for every generation. Default is `FALSE`

Value

A list of SpatRasters

Examples

```
## Not run:
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(4651704, 4651704, 4654475, 4654475, 4651704),
    c(2692925, 2694746, 2694746, 2692925, 2692925)
  ))),
  crs = 3035
```

```
))
Polygon_wgs84 <- sf::st_transform(Polygon1, st_crs(4326))
srtm <- elevatr::get_elev_raster(locations = Polygon_wgs84, z = 11)
res <- terrain_model(srtm, Polygon1)

## End(Not run)
```

`trimton` *Adjust the amount of turbines per windfarm*

Description

Adjust the mutated individuals to the required amount of turbines.

Usage

```
trimton(mut, nturb, allparks, nGrids, trimForce, seed)
```

Arguments

`mut`	A binary matrix with the mutated individuals
`nturb`	A numeric value indicating the amount of required turbines
`allparks`	A data.frame consisting of all individuals of the current generation
`nGrids`	A numeric value indicating the total amount of grid cells
`trimForce`	If `TRUE` the algorithm will use a probabilistic approach to correct the windfarms to the desired amount of turbines. If `FALSE` the adjustment will be random. Default is `FALSE`
`seed`	Set a seed for comparability. Default is NULL

Value

Returns a binary matrix with the correct amount of turbines per individual

See Also

Other Genetic Algorithm Functions: `crossover()`, `fitness()`, `genetic_algorithm()`, `init_population()`, `mutation()`, `selection()`

Examples

```
## Create a random rectangular shapefile
library(sf)
Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(0, 0, 2000, 2000, 0),
    c(0, 2000, 2000, 0, 0)
  ))),
```

```
  crs = 3035
))

## Create a uniform and unidirectional wind data.frame and plots the
## resulting wind rose
## Uniform wind speed and single wind direction
data.in <- as.data.frame(cbind(ws = 12, wd = 0))

## Calculate a Grid and an indexed data.frame with coordinates and grid cell Ids.
Grid1 <- grid_area(shape = Polygon1, size = 200, prop = 1)
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- init_population(Grid, 10, 20)
wind <- as.data.frame(cbind(ws = 12, wd = 0))
wind <- list(wind, probab = 100)
fit <- fitness(
  selection = startsel, referenceHeight = 100, RotorHeight = 100,
  SurfaceRoughness = 0.3, Polygon = Polygon1, resol1 = 200, rot = 20,
  dirspeed = wind, srtm_crop = "", topograp = FALSE, cclRaster = ""
)
allparks <- do.call("rbind", fit)
## SELECTION
## print the amount of Individuals selected.
## Check if the amount of Turbines is as requested.
selec6best <- selection(fit, Grid, 2, TRUE, 6, "VAR")
selec6best <- selection(fit, Grid, 2, TRUE, 6, "FIX")
selec6best <- selection(fit, Grid, 4, FALSE, 6, "FIX")
## CROSSOVER
## u determines the amount of crossover points,
## crossPart determines the method used (Equal/Random),
## uplimit is the maximum allowed permutations
crossOut <- crossover(selec6best, 2, uplimit = 300, crossPart = "RAN")
crossOut <- crossover(selec6best, 7, uplimit = 500, crossPart = "RAN")
crossOut <- crossover(selec6best, 3, uplimit = 300, crossPart = "EQU")
## MUTATION
## Variable Mutation Rate is activated if more than 2 individuals represent
## the current best solution.
mut <- mutation(a = crossOut, p = 0.3, NULL)
## TRIMTON
## After Crossover and Mutation, the amount of turbines in a windpark change and have to be
## corrected to the required amount of turbines.
mut1 <- trimton(
  mut = mut, nturb = 10, allparks = allparks, nGrids = AmountGrids,
  trimForce = FALSE
)
colSums(mut)
colSums(mut1)
```

`turbine_influences` *Find potentially influencing turbines*

Description

Find all turbines that could potentially influence another turbine and save them to a list.

Usage

```
turbine_influences(t, wnkl, dist, polYgon, dirct, plotAngles = FALSE)
```

Arguments

`t`	A data.frame of the current individual with X and Y coordinates
`wnkl`	The angle from which wake influences are considered to be negligible
`dist`	A numeric value indicating the distance, after which the wake effects are considered to be eliminated.
`polYgon`	A shapefile representing the considered area
`dirct`	A numeric value indicating the current wind direction
`plotAngles`	A logical variable, which is used to plot the distances and angles. Default is `FALSE`

Value

Returns a list of all individuals of the current generation which could potentially influence other turbines. List includes the relevant coordinates, the distances and angles in between and assigns the Point ID.

See Also

Other Wind Energy Calculation Functions: `barometric_height()`, `calculate_energy()`, `circle_intersection()`, `get_dist_angles()`

Examples

```
## Exemplary input Polygon with 2km x 2km:
library(sf)

Polygon1 <- sf::st_as_sf(sf::st_sfc(
  sf::st_polygon(list(cbind(
    c(0, 0, 2000, 2000, 0),
    c(0, 2000, 2000, 0, 0)
  ))),
  crs = 3035
))

t <- st_coordinates(st_sample(Polygon1, 10))
t <- cbind(t, "Z" = 1)
wnkl <- 20
```

```
dist <- 100000
dirct <- 0

res <- turbine_influences(t, wnkl, dist, Polygon1, dirct, plotAngles = TRUE)
```

`windata_format` *Transform Winddata*

Description

Helper Function, which transforms winddata to an acceptable format

Usage

```
windata_format(df)
```

Arguments

`df`	The wind data with speeds, direction and optionally a probability column. If not assigned, it will be calculated

Value

A list of windspeed and probabilities

See Also

Other Helper Functions: `get_grids()`, `grid_area()`, `hexa_area()`, `isSpatial()`, `permutations()`, `readinteger()`, `readintegerSel()`, `splitAt()`

Examples

```
wind_df <- data.frame(
  ws = c(12, 30, 45),
  wd = c(0, 90, 150),
  probab = 30:32
)
windata_format(wind_df)

wind_df <- data.frame(
  speed = c(12, 30, 45),
  direction = c(90, 90, 150),
  probab = c(10, 20, 60)
)
windata_format(wind_df)

wind_df <- data.frame(
  speed = c(12, 30, 45),
  direction = c(400, 90, 150)
```

```
)
windata_format(wind_df)
```

Index