Package 'spsann'

April 29, 2019

Type Package

Title Optimization of Sample Configurations using Spatial Simulated Annealing

Version 2.2.0

Date 2019-04-28

Description
Methods to optimize sample configurations using spatial simulated annealing. Multiple objective functions are implemented for various purposes, such as variogram estimation, spatial trend estimation
and spatial interpolation. A general purpose spatial simulated annealing function enables the user to
define his/her own objective function. Solutions for augmenting existing sample configurations and solving
multi-objective optimization problems are available as well.

License GPL (>= 2)

Imports methods, pedometrics, Rcpp, sp, SpatialTools

Suggests gstat, tcltk, knitr

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 6.1.1

VignetteBuilder knitr

URL https://github.com/samuel-rosa/spsann/

BugReports https://github.com/samuel-rosa/spsann/issues/

Language en-GB

NeedsCompilation yes

Author Alessandro Samuel-Rosa [aut, cre]
(<https://orcid.org/0000-0003-0877-1320>),
Lucia Helena Cunha dos Anjos [ths]
(<https://orcid.org/0000-0003-0063-3521>),
Gustavo de Mattos Vasques [ths],

Gerard B M Heuvelink [ths] (<https://orcid.org/0000-0003-0959-9358>),
Dick Brus [ctb] (<https://orcid.org/0000-0003-2194-4783>),
Richard Murray Lark [ctb] (<https://orcid.org/0000-0003-2571-8521>),
Edzer Pebesma [ctb] (<https://orcid.org/0000-0001-8049-7069>),
Jon Skoien [ctb],
Joshua French [ctb],
Pierre Roudier [ctb]

Maintainer Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

Repository CRAN

Date/Publication 2019-04-29 12:20:03 UTC

R topics documented:

---

spsann-package *The spsann Package*

Description

Optimization of sample configurations using spatial simulated annealing

Introduction

spsann is a package for the optimization of spatial sample configurations using spatial simulated annealing. It includes multiple objective functions to optimize spatial sample configurations for various purposes such as variogram estimation, spatial trend estimation, and spatial interpolation. Most of the objective functions were designed to optimize spatial sample configurations when a) multiple spatial variables must be modelled, b) we know very little about the model of spatial variation of those variables, and c) sampling is limited to a single phase.

Spatial simulated annealing is a well known method with widespread use to solve combinatorial optimization problems in the environmental sciences. This is mainly due to its robustness against local optima and easiness to implement. In short, the algorithm consists of randomly changing the spatial location of a candidate sampling point at a time and evaluating if the resulting spatial sample configuration is *better* than the previous one with regard to the chosen quality criterion, i.e. an objective function. Sometimes a *worse* spatial sample configuration is accepted so that the algorithm is able to scape from local optima solutions, i.e. those spatial sample configurations that are too good and appear to early in the optimization to be true. The chance of accepting a *worse* spatial sample configuration reduces as the optimization proceeds so that we can get very close to the *optimum* spatial sample configuration.

spsann also combines multiple objective functions so that spatial sample configurations can be optimized regarding more than one modelling objective. Combining multiple objective functions gives rise to a multi-objective combinatorial optimization problem (MOCOP). A MOCOP usually has multiple possible solutions. **spsann** finds a single solution by aggregating the objective functions using the weighted-sum method. With this method the relative importance of every objective function can be specified at the beginning of the optimization so that their relative influence on the resulting optimized spatial sample configuration can be different. But this requires the objective functions first to be scaled to the same approximate range of values. The upper-lower bound approach is used for that end. In this approach, every objective function is scaled using as reference the respective minimum and maximum attainable objective function values, also known as the Pareto minimum and maximum.

Package Structure

spsann has a very simple structure composed of three families of functions. The first is the family of `optim` functions. These are the functions that include the spatial simulated annealing algorithm, that is, the functions that perform the optimization regarding the chosen quality criterion (objective function). Every `optim` function is named after the objective function used as quality criterion. For example, the quality criterion used by `optimMSSD` is the *mean squared shortest distance* (MSSD) between sample and prediction points. As the example shows, the name of the `optim` functions is composed of the string `'optim'` followed by a suffix that indicates the respective objective function. In the example this is `'MSSD'`.

There currently are nine function in the `optim` family: `optimACDC`, `optimCLHS`, `optimCORR`, `optimDIST`, `optimMSSD`, `optimMKV`, `optimPPL`, `optimSPAN`, and `optimUSER`. The latter is a general purpose function that enables to user to define his/her own objective function and plug it in the spatial simulated annealing algorithm.

The second family of functions is the `obj` family. This family of functions is used to return the current objective function value of a spatial sample configuration. Like the family of `optim` functions, the name of the `obj` functions is composed of the string `'obj'` plus a suffix that indicates the objective function being used. For example, `objMSSD` computes the value of the mean squared shortest distance between sample and prediction points of any spatial sample configuration. Accordingly, there is a `obj` function for every `optim` function, except for `optimUSER`. A ninth `obj` function, `objSPSANN`, returns the objective function value at any point of the optimization, irrespective of the objective function used.

The third family of functions implemented in **spsann** corresponds to a set of auxiliary functions. These auxiliary functions can be used for several purposes, such as organizing the information needed to feed an `optim` function, retrieving information from an object of class `OptimizedSampleConfiguration`,

i.e. an object containing an optimized sample configuration, generating plots of the spatial distribution an optimized sample configuration, and so on. These functions are named after the purpose for which they have been designed. For example: `countPPL`, `minmaxPareto`, `scheduleSPSANN`, `spJitter`, and `plot`.

Despite **spsann** functions are classified into three general family of functions defined according to the purpose for which they were designed, the documentation is constructed with regard to the respective objective functions. For example, every **spsann** function that uses as quality criterion the MSSD is documented in the same documentation page. The exception are the auxiliary functions, that generally are documented separately.

Support

spsann was initially developed as part of the PhD research project entitled 'Contribution to the Construction of Models for Predicting Soil Properties', developed by Alessandro Samuel-Rosa under the supervision of Lúcia Helena Cunha dos Anjos <lanjos@ufrrj.br> (Universidade Federal Rural do Rio de Janeiro, Brazil), Gustavo de Mattos Vasques <gustavo.vasques@embrapa.br> (Embrapa Solos, Brazil), and Gerard B. M. Heuvelink <gerard.heuvelink@wur.nl> (ISRIC – World Soil Information, the Netherlands). The project was supported from March/2012 to February/2016 by the CAPES Foundation, Ministry of Education of Brazil, and the CNPq Foundation, Ministry of Science and Technology of Brazil.

Contributors

Some of the solutions used to build **spsann** were found in the source code of other R-packages and scripts developed and published by other researchers. For example, the original skeleton of the optimization functions was adopted from the **intamapInteractive** package with the approval of the package authors, Edzer Pebesma <edzer.pebesma@uni-muenster.de> and Jon Skoien <jon.skoien@gmail.com>. The current skeleton is based on the later adoption of several solutions implemented in the script developed and published by Murray Lark <mlark@bgs.ac.uk> as part of a short course ('Computational tools to optimize spatial sampling') offered for the first time at the 2015 EGU General Assembly in Vienna, Austria.

A few small solutions were adopted from the packages **SpatialTools**, authored by Joshua French <joshua.french@ucdenver.edu>, **clhs**, authored by Pierre Roudier <roudierp@landcareresearch.co.nz>, and **spcosa**, authored by Dennis Walvoort <dennis.Walvoort@wur.nl>, Dick Brus <dick.brus@wur.nl>, and Jaap de Gruijter <Jaap.degruijter@wur.nl>.

Major conceptual contributions were made by Gerard Heuvelink <gerard.heuvelink@wur.nl>, Dick Brus <dick.brus@wur.nl>, Murray Lark <mlark@bgs.ac.uk>, and Edzer Pebesma <edzer.pebesma@uni-muenster.

Author(s)

Author and Maintainer: Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>.

`minmaxPareto` *Pareto minimum and maximum values*

Description

Compute the minimum and maximum attainable values of the objective functions that compose a multi-objective combinatorial optimization problem.

Usage

```
minmaxPareto(osc, candi, covars)
```

Arguments

`osc`	A list of objects of class `OptimizedSampleConfiguration` (OSC). Each OSC of the list must be named after the objective function with which it has been optimized. For example, `osc = list(CORR = osc_corr, DIST = osc_dist)`.
`candi`	Data frame or matrix with the candidate locations for the jittered points. `candi` must have two columns in the following order: `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.
`covars`	Data frame or matrix with the covariates in the columns.

Details

Multi-objective combinatorial optimization problems: A method of solving a multi-objective combinatorial optimization problem (MOCOP) is to aggregate the objective functions into a single *utility function*. In **spsann**, the aggregation is performed using the *weighted sum method*, which incorporates in the weights the preferences of the user regarding the relative importance of each objective function.

The weighted sum method is affected by the relative magnitude of the different function values. The objective functions implemented in **spsann** have different units and orders of magnitude. The consequence is that the objective function with the largest values may have a numerical dominance during the optimization. In other words, the weights may not express the true preferences of the user, resulting that the meaning of the utility function becomes unclear because the optimization will favour the objective function which is numerically dominant.

A reasonable solution to avoid the numerical dominance of any objective function is to scale the objective functions so that they are constrained to the same approximate range of values. Several function-transformation methods can be used for this end and **spsann** has four of them available.

The *upper-lower-bound approach* requires the user to inform the maximum (nadir point) and minimum (utopia point) absolute function values. The resulting function values will always range between 0 and 1.

The *upper-bound approach* requires the user to inform only the nadir point, while the utopia point is set to zero. The upper-bound approach for transformation aims at equalizing only the upper bounds of the objective functions. The resulting function values will always be smaller than or equal to 1.

In most cases, the absolute maximum and minimum values of an objective function cannot be calculated exactly. If the user is uncomfortable with guessing the nadir and utopia points, there an option for using *numerical simulations*. It consists of computing the function value for many random system configurations. The mean function value obtained over multiple simulations is used to set the nadir point, while the the utopia point is set to zero. This approach is similar to the upper-bound approach, but the function values will have the same orders of magnitude only at the starting point of the optimization. Function values larger than one are likely to occur during the optimization. We recommend the user to avoid this approach whenever possible because the effect of the starting configuration on the optimization as a whole usually is insignificant or arbitrary.

The *upper-lower-bound approach* with the minimum and maximum *attainable* values of the objective functions that compose the MOCOP, also known as the *Pareto minimum and maximum values*, is the most elegant solution to scale the objective functions. However, it is the most time consuming. It works as follows:

1. Optimize a sample configuration with respect to each objective function that composes the MOCOP;
2. Compute the function value of every objective function that composes the MOCOP for every optimized sample configuration;
3. Record the minimum and maximum absolute function values attained for each objective function that composes the MOCOP – these are the Pareto minimum and maximum.

For example, consider **ACDC**, a MOCOP composed of two objective functions: **CORR** and **DIST**. The minimum absolute attainable value of **CORR** is obtained when the sample configuration is optimized with respect to only **CORR**, i.e. when the evaluator and generator objective functions are the same (see the intersection between the second line and second column in the table below). This is the Pareto minimum of **CORR**. It follows that the maximum absolute attainable value of **CORR** is obtained when the sample configuration is optimized with regard to only **DIST**, i.e. when the evaluator function is difference from the generator function (see the intersection between the first row and the second column in the table below). This is the Pareto maximum of **CORR**. The same logic applies for finding the Pareto minimum and maximum of **DIST**.

Evaluator	*Generator*	
	DIST	CORR
DIST	0.5	8.6
CORR	6.4	0.3

Value

A data frame with the Pareto minimum and maximum values.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Arora, J. *Introduction to optimum design*. Waltham: Academic Press, p. 896, 2011.

Marler, R. T.; Arora, J. S. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, v. 26, p. 369-395, 2004.

Marler, R. T.; Arora, J. S. Function-transformation methods for multi-objective optimization. *Engineering Optimization*, v. 37, p. 551-570, 2005.

Marler, R. T.; Arora, J. S. The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, v. 41, p. 853-862, 2009.

See Also

optimACDC, SPAN

Examples

```
## Not run:
# This example takes more than 5 seconds
require(sp)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
covars <- meuse.grid[, c(1, 2)]

# CORR
schedule <- scheduleSPSANN(initial.acceptance = 0.1, chains = 1,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)
set.seed(2001)
osc_corr <- optimCORR(points = 10, candi = candi, covars = covars,
                      schedule = schedule)

# DIST
set.seed(2001)
osc_dist <- optimDIST(points = 10, candi = candi, covars = covars,
                      schedule = schedule)

# PPL
set.seed(2001)
osc_ppl <- optimPPL(points = 10, candi = candi, schedule = schedule)

# MSSD
set.seed(2001)
osc_mssd <- optimMSSD(points = 10, candi = candi, schedule = schedule)

# Pareto
pareto <- minmaxPareto(osc = list(DIST = osc_dist, CORR = osc_corr,
                                  PPL = osc_ppl, MSSD = osc_mssd),
                       candi = candi, covars = covars)
pareto

## End(Not run)
```

objSPSANN *Auxiliary tools*

Description

Auxiliary tools used in the optimization of sample configurations using spatial simulated annealing.

Usage

```
objSPSANN(osc, at = "end", n = 1)
```

Arguments

osc	Object of class OptimizedSampleConfiguration.
at	Point of the optimization at which the energy state should be returned. Available options: "start", for the start, and "end", for the end of the optimization. Defaults to at = "end".
n	Number of instances that should be returned. Defaults to n = 1.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

optimACDC *Optimization of sample configurations for spatial trend identification and estimation (III)*

Description

Optimize a sample configuration for spatial trend identification and estimation. An utility function U is defined so that the sample reproduces the bivariate association/correlation between the covariates, as well as their marginal distribution (**ACDC**). The utility function is obtained aggregating two objective functions: **CORR** and **DIST**.

Usage

```
optimACDC(points, candi, covars, strata.type = "area",
  use.coords = FALSE, schedule = scheduleSPSANN(), plotit = FALSE,
  track = FALSE, boundary, progress = "txt", verbose = FALSE,
  weights, nadir = list(sim = NULL, seeds = NULL, user = NULL, abs =
  NULL), utopia = list(user = NULL, abs = NULL))

objACDC(points, candi, covars, strata.type = "area",
  use.coords = FALSE, weights, nadir = list(sim = NULL, seeds = NULL,
  user = NULL, abs = NULL), utopia = list(user = NULL, abs = NULL))
```

Arguments

points	Integer value, integer vector, data frame or matrix, or list. • Integer value. The number of points. These points will be randomly sampled from candi to form the starting sample configuration. • Integer vector. The row indexes of candi that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points. • Data frame or matrix. An object with three columns in the following order: [, "id"], the row indexes of candi that correspond to each point, [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates. • List. An object with two named sub-arguments: fixed, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and free, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.
candi	Data frame or matrix with the candidate locations for the jittered points. candi must have two columns in the following order: [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates.
covars	Data frame or matrix with the covariates in the columns.
strata.type	(Optional) Character value setting the type of stratification that should be used to create the marginal sampling strata (or factor levels) for the numeric covariates. Available options are "area", for equal-area, and "range", for equal-range. Defaults to strata.type = "area".
use.coords	(Optional) Logical value. Should the spatial x- and y-coordinates be used as covariates? Defaults to use.coords = FALSE.
schedule	List with 11 named sub-arguments defining the control parameters of the cooling schedule. See scheduleSPSANN.
plotit	(Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to plotit = FALSE.
track	(Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If track = FALSE (the default), only the starting and ending energy states are returned along with the results.
boundary	(Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and plotit = TRUE, boundary is estimated from candi.
progress	(Optional) Type of progress bar that should be used, with options "txt", for a text progress bar in the R console, "tk", to put up a Tk progress bar widget, and NULL to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to progress = "txt".
verbose	(Optional) Logical for printing messages about the progress of the optimization. Defaults to verbose = FALSE.

`weights` List with named sub-arguments. The weights assigned to each one of the objective functions that form the multi-objective combinatorial optimization problem. They must be named after the respective objective function to which they apply. The weights must be equal to or larger than 0 and sum to 1.

`nadir` List with named sub-arguments. Three options are available: 1) `sim` – the number of simulations that should be used to estimate the nadir point, and `seeds` – vector defining the random seeds for each simulation; 2) `user` – a list of user-defined nadir values named after the respective objective functions to which they apply; 3) `abs` – logical for calculating the nadir point internally (experimental).

`utopia` List with named sub-arguments. Two options are available: 1) `user` – a list of user-defined values named after the respective objective functions to which they apply; 2) `abs` – logical for calculating the utopia point internally (experimental).

Details

The help page of `minmaxPareto` contains details on how **spsann** solves the multi-objective combinatorial optimization problem of finding a globally optimum sample configuration that meets multiple, possibly conflicting, sampling objectives.

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of `spJitter`.

Visit the help pages of `optimCORR` and `optimDIST` to see the details of the objective functions that compose **ACDC**.

Value

`optimACDC` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

`objACDC` returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

This function was derived with modifications from the method known as the *conditioned Latin Hypercube sampling* originally proposed by Minasny and McBratney (2006), and implemented in the R-package **clhs** by Pierre Roudier.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Minasny, B.; McBratney, A. B. A conditioned Latin hypercube method for sampling in the presence of ancillary information. *Computers & Geosciences*, v. 32, p. 1378-1388, 2006.

Minasny, B.; McBratney, A. B. Conditioned Latin Hypercube Sampling for calibrating soil sensor data to soil properties. Chapter 9. Viscarra Rossel, R. A.; McBratney, A. B.; Minasny, B. (Eds.) *Proximal Soil Sensing*. Amsterdam: Springer, p. 111-119, 2010.

Roudier, P.; Beaudette, D.; Hewitt, A. A conditioned Latin hypercube sampling algorithm incorporating operational constraints. *5th Global Workshop on Digital Soil Mapping*. Sydney, p. 227-231, 2012.

See Also

cramer

Examples

```
data(meuse.grid, package = "sp")
candi <- meuse.grid[1:1000, 1:2]
nadir <- list(sim = 10, seeds = 1:10)
utopia <- list(user = list(DIST = 0, CORR = 0))
covars <- meuse.grid[1:1000, 5]
schedule <- scheduleSPSANN(
  chains = 1, initial.temperature = 5, x.max = 1540, y.max = 2060,
  x.min = 0, y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimACDC(
  points = 10, candi = candi, covars = covars, nadir = nadir, use.coords = TRUE,
  utopia = utopia, schedule = schedule, weights = list(DIST = 1/2, CORR = 1/2))
objSPSANN(res) - objACDC(
  points = res, candi = candi, covars = covars, use.coords = TRUE, nadir = nadir,
  utopia = utopia, weights = list(DIST = 1/2, CORR = 1/2))
```

---

optimCLHS *Optimization of sample configurations for spatial trend identification and estimation (IV)*

Description

Optimize a sample configuration for spatial trend identification and estimation using the method proposed by Minasny and McBratney (2006), known as the conditioned Latin hypercube sampling. An utility function U is defined so that the sample reproduces the marginal distribution and correlation matrix of the numeric covariates, and the class proportions of the factor covariates (**CLHS**). The utility function is obtained aggregating three objective functions: **O1**, **O2**, and **O3**.

Usage

```
optimCLHS(points, candi, covars, use.coords = FALSE,
  clhs.version = c("paper", "fortran", "update"),
  schedule = scheduleSPSANN(), plotit = FALSE, track = FALSE,
  boundary, progress = "txt", verbose = FALSE, weights)

objCLHS(points, candi, covars, use.coords = FALSE,
  clhs.version = c("paper", "fortran", "update"), weights)
```

Arguments

`points`	Integer value, integer vector, data frame or matrix, or list. • Integer value. The number of points. These points will be randomly sampled from `candi` to form the starting sample configuration. • Integer vector. The row indexes of `candi` that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points. • Data frame or matrix. An object with three columns in the following order: `[, "id"]`, the row indexes of `candi` that correspond to each point, `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates. • List. An object with two named sub-arguments: `fixed`, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and `free`, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.
`candi`	Data frame or matrix with the candidate locations for the jittered points. `candi` must have two columns in the following order: `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.
`covars`	Data frame or matrix with the covariates in the columns.
`use.coords`	(Optional) Logical value. Should the spatial x- and y-coordinates be used as covariates? Defaults to `use.coords = FALSE`.
`clhs.version`	(Optional) Character value setting the CLHS version that should be used. Available options are: `"paper"`, for the formulations of **O1**, **O2**, and **O3** as presented in the original paper by Minasny and McBratney (2006); `"fortran"`, for the formulations of **O1** and **O3** that include a scaling factor as implemented in the late Fortran code by Budiman Minasny (ca. 2015); and `"update"`, for formulations of **O1**, **O2**, and **O3** that include the modifications proposed the authors of this package in 2018 (see below). Defaults to `clhs.version = "paper"`.
`schedule`	List with 11 named sub-arguments defining the control parameters of the cooling schedule. See `scheduleSPSANN`.
`plotit`	(Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to `plotit = FALSE`.

`track` (Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If `track = FALSE` (the default), only the starting and ending energy states are returned along with the results.

`boundary` (Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and `plotit = TRUE`, `boundary` is estimated from `candi`.

`progress` (Optional) Type of progress bar that should be used, with options `"txt"`, for a text progress bar in the R console, `"tk"`, to put up a Tk progress bar widget, and `NULL` to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to `progress = "txt"`.

`verbose` (Optional) Logical for printing messages about the progress of the optimization. Defaults to `verbose = FALSE`.

`weights` List with named sub-arguments. The weights assigned to each one of the objective functions that form the multi-objective combinatorial optimization problem. They must be named after the respective objective function to which they apply. The weights must be equal to or larger than 0 and sum to 1.

Details

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of `spJitter`.

Marginal sampling strata: Reproducing the marginal distribution of the numeric covariates depends upon the definition of marginal sampling strata. *Equal-area* marginal sampling strata are defined using the sample quantiles estimated with `quantile` using a continuous function (`type = 7`), that is, a function that interpolates between existing covariate values to estimate the sample quantiles. This is the procedure implemented in the original method of Minasny and McBratney (2006), which creates breakpoints that do not occur in the population of existing covariate values. Depending on the level of discretization of the covariate values, that is, how many significant digits they have, this can create repeated breakpoints, resulting in empty marginal sampling strata. The number of empty marginal sampling strata will ultimately depend on the frequency distribution of the covariate and on the number of sampling points. The effect of these features on the spatial modelling outcome still is poorly understood.

Correlation between numeric covariates: The *correlation* between two numeric covariates is measured using the sample Pearson's *r*, a descriptive statistic that ranges from -1 to +1. This statistic is also known as the sample linear correlation coefficient. The effect of ignoring the correlation among factor covariates and between factor and numeric covariates on the spatial modelling outcome still is poorly understood.

Multi-objective combinatorial optimization: A method of solving a multi-objective combinatorial optimization problem (MOCOP) is to aggregate the objective functions into a single utility function *U*. In the **spsann** package, as in the original implementation of the CLHS by Minasny and McBratney (2006), the aggregation is performed using the **weighted sum method**, which uses weights to incorporate the **a priori** preferences of the user about the relative importance of each objective function. When the user has no preference, the objective functions receive equal weights.

The weighted sum method is affected by the relative magnitude of the different objective function values. The objective functions implemented in `optimCLHS` have different units and orders of magnitude. The consequence is that the objective function with the largest values, generally **O1**, may have a numerical dominance during the optimization. In other words, the weights may not express the true preferences of the user, resulting that the meaning of the utility function becomes unclear because the optimization will likely favour the objective function which is numerically dominant.

An efficient solution to avoid numerical dominance is to scale the objective functions so that they are constrained to the same approximate range of values, at least in the end of the optimization. In the original implementation of the CLHS by Minasny and McBratney (2006), `clhs.version = "paper"`, `optimCLHS` uses the naive aggregation method, which ignores that the three objective functions have different units and orders of magnitude. In a 2015 Fortran implementation of the CLHS, `clhs.version = "fortran"`, scaling factors were included to make the values of the three objective function more comparable. The effect of ignoring the need to scale the objective functions, or using arbitrary scaling factors, on the spatial modelling outcome still is poorly understood. Thus, an updated version of **O1**, **O2**, and **O3** has been implemented in the **spsann** package. The need formulation aim at making the values returned by the objective functions more comparable among themselves without having to resort to arbitrary scaling factors. The effect of using these new formulations have not been tested yet.

Value

`optimCLHS` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

`objCLHS` returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

The (only?) difference of `optimCLHS` to the original Fortran implementation of Minasny and McBratney (2006), and to the `clhs` function implemented in the former **clhs** package by Pierre Roudier, is the annealing schedule.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Minasny, B.; McBratney, A. B. A conditioned Latin hypercube method for sampling in the presence of ancillary information. *Computers & Geosciences*, v. 32, p. 1378-1388, 2006.

Minasny, B.; McBratney, A. B. Conditioned Latin Hypercube Sampling for calibrating soil sensor data to soil properties. Chapter 9. Viscarra Rossel, R. A.; McBratney, A. B.; Minasny, B. (Eds.) *Proximal Soil Sensing*. Amsterdam: Springer, p. 111-119, 2010.

Roudier, P.; Beaudette, D.; Hewitt, A. A conditioned Latin hypercube sampling algorithm incorporating operational constraints. *5th Global Workshop on Digital Soil Mapping*. Sydney, p. 227-231, 2012.

See Also

optimACDC

Examples

```
data(meuse.grid, package = "sp")
candi <- meuse.grid[1:1000, 1:2]
covars <- meuse.grid[1:1000, 5]
schedule <- scheduleSPSANN(
  chains = 1, initial.temperature = 20, x.max = 1540, y.max = 2060,
  x.min = 0, y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimCLHS(
  points = 10, candi = candi, covars = covars, use.coords = TRUE,
  clhs.version = "fortran", weights = list(O1 = 0.5, O3 = 0.5), schedule = schedule)
objSPSANN(res) - objCLHS(
  points = res, candi = candi, covars = covars, use.coords = TRUE,
  clhs.version = "fortran", weights = list(O1 = 0.5, O3 = 0.5))
```

---

optimCORR — *Optimization of sample configurations for spatial trend identification and estimation (I)*

Description

Optimize a sample configuration for spatial trend identification and estimation. A criterion is defined so that the sample reproduces the bivariate association/correlation between the covariates (**CORR**).

Usage

```
optimCORR(points, candi, covars, strata.type = "area",
  use.coords = FALSE, schedule = scheduleSPSANN(), plotit = FALSE,
  track = FALSE, boundary, progress = "txt", verbose = FALSE)

objCORR(points, candi, covars, strata.type = "area",
  use.coords = FALSE)
```

Arguments

`points` Integer value, integer vector, data frame or matrix, or list.

- Integer value. The number of points. These points will be randomly sampled from `candi` to form the starting sample configuration.

- Integer vector. The row indexes of `candi` that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points.
- Data frame or matrix. An object with three columns in the following order: `[, "id"]`, the row indexes of `candi` that correspond to each point, `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.
- List. An object with two named sub-arguments: `fixed`, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and `free`, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.

`candi` Data frame or matrix with the candidate locations for the jittered points. `candi` must have two columns in the following order: `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.

`covars` Data frame or matrix with the covariates in the columns.

`strata.type` (Optional) Character value setting the type of stratification that should be used to create the marginal sampling strata (or factor levels) for the numeric covariates. Available options are `"area"`, for equal-area, and `"range"`, for equal-range. Defaults to `strata.type = "area"`.

`use.coords` (Optional) Logical value. Should the spatial x- and y-coordinates be used as covariates? Defaults to `use.coords = FALSE`.

`schedule` List with 11 named sub-arguments defining the control parameters of the cooling schedule. See `scheduleSPSANN`.

`plotit` (Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to `plotit = FALSE`.

`track` (Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If `track = FALSE` (the default), only the starting and ending energy states are returned along with the results.

`boundary` (Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and `plotit = TRUE`, `boundary` is estimated from `candi`.

`progress` (Optional) Type of progress bar that should be used, with options `"txt"`, for a text progress bar in the R console, `"tk"`, to put up a Tk progress bar widget, and `NULL` to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to `progress = "txt"`.

`verbose` (Optional) Logical for printing messages about the progress of the optimization. Defaults to `verbose = FALSE`.

Details

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of `spJitter`.

Association/Correlation between covariates: The *correlation* between two numeric covariates is measured using the Pearson's *r*, a descriptive statistic that ranges from -1 to $+1$. This statistic is also known as the linear correlation coefficient.

When the set of covariates includes factor covariates, all numeric covariates are transformed into factor covariates. The factor levels are defined using the marginal sampling strata created from one of the two methods available (equal-area or equal-range strata).

The *association* between two factor covariates is measured using the Cramér's *V*, a descriptive statistic that ranges from 0 to $+1$. The closer to $+1$ the Cramér's *V* is, the stronger the association between two factor covariates.

The main weakness of using the Cramér's *V* is that, while the Pearson's *r* shows the degree and direction of the association between two covariates (negative or positive), the Cramér's *V* only measures the degree of association (weak or strong). The effect of replacing the Pearson's *r* with the Cramér's *V* on the spatial modelling outcome still is poorly understood.

Value

`optimCORR` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

`objCORR` returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Cramér, H. *Mathematical methods of statistics*. Princeton: Princeton University Press, p. 575, 1946.

Everitt, B. S. *The Cambridge dictionary of statistics*. Cambridge: Cambridge University Press, p. 432, 2006.

See Also

`cramer`, `optimACDC`

Examples

```
data(meuse.grid, package = "sp")
candi <- meuse.grid[1:1000, 1:2]
covars <- meuse.grid[1:1000, 5]
```

```
schedule <- scheduleSPSANN(
  initial.temperature = 5, chains = 1, x.max = 1540, y.max = 2060,
  x.min = 0, y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimCORR(
  points = 10, candi = candi, covars = covars, use.coords = TRUE,
  schedule = schedule)
objSPSANN(res) - objCORR(
  points = res, candi = candi, covars = covars, use.coords = TRUE)
```

---

optimDIST *Optimization of sample configurations for spatial trend identification and estimation (II)*

Description

Optimize a sample configuration for spatial trend identification and estimation. A criterion is defined so that the sample reproduces the marginal distribution of the covariates (**DIST**).

Usage

```
optimDIST(points, candi, covars, strata.type = "area",
  use.coords = FALSE, schedule = scheduleSPSANN(), plotit = FALSE,
  track = FALSE, boundary, progress = "txt", verbose = FALSE)

objDIST(points, candi, covars, strata.type = "area",
  use.coords = FALSE)
```

Arguments

`points` Integer value, integer vector, data frame or matrix, or list.

- Integer value. The number of points. These points will be randomly sampled from `candi` to form the starting sample configuration.
- Integer vector. The row indexes of `candi` that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points.
- Data frame or matrix. An object with three columns in the following order: `[, "id"]`, the row indexes of `candi` that correspond to each point, `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.
- List. An object with two named sub-arguments: `fixed`, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and `free`, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.

`candi` Data frame or matrix with the candidate locations for the jittered points. `candi` must have two columns in the following order: `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.

covars	Data frame or matrix with the covariates in the columns.
strata.type	(Optional) Character value setting the type of stratification that should be used to create the marginal sampling strata (or factor levels) for the numeric covariates. Available options are "area", for equal-area, and "range", for equal-range. Defaults to strata.type = "area".
use.coords	(Optional) Logical value. Should the spatial x- and y-coordinates be used as covariates? Defaults to use.coords = FALSE.
schedule	List with 11 named sub-arguments defining the control parameters of the cooling schedule. See scheduleSPSANN.
plotit	(Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to plotit = FALSE.
track	(Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If track = FALSE (the default), only the starting and ending energy states are returned along with the results.
boundary	(Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and plotit = TRUE, boundary is estimated from candi.
progress	(Optional) Type of progress bar that should be used, with options "txt", for a text progress bar in the R console, "tk", to put up a Tk progress bar widget, and NULL to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to progress = "txt".
verbose	(Optional) Logical for printing messages about the progress of the optimization. Defaults to verbose = FALSE.

Details

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of spJitter.

Marginal distribution of covariates: Reproducing the marginal distribution of the numeric covariates depends upon the definition of marginal sampling strata. These marginal sampling strata are also used to define the factor levels of all numeric covariates that are passed together with factor covariates. Two types of marginal sampling strata can be used: *equal-area* and *equal-range*.

Equal-area marginal sampling strata are defined using the sample quantiles estimated with quantile using a discontinuous function (type = 3). Using a discontinuous function avoids creating breakpoints that do not occur in the population of existing covariate values.

Depending on the level of discretization of the covariate values, quantile produces repeated breakpoints. A breakpoint will be repeated if that value has a relatively high frequency in the population of covariate values. The number of repeated breakpoints increases with the number of marginal sampling strata. Repeated breakpoints result in empty marginal sampling strata. To avoid this, only the unique breakpoints are used.

Equal-range marginal sampling strata are defined by breaking the range of covariate values into pieces of equal size. Depending on the level of discretization of the covariate values, this method creates breakpoints that do not occur in the population of existing covariate values. Such breakpoints are replaced with the nearest existing covariate value identified using Euclidean distances.

Like the equal-area method, the equal-range method can produce empty marginal sampling strata. The solution used here is to merge any empty marginal sampling strata with the closest non-empty marginal sampling strata. This is identified using Euclidean distances as well.

The approaches used to define the marginal sampling strata result in each numeric covariate having a different number of marginal sampling strata, some of them with different area/size. Because the goal is to have a sample that reproduces the marginal distribution of the covariate, each marginal sampling strata will have a different number of sample points. The wanted distribution of the number of sample points per marginal strata is estimated empirically as the proportion of points in the population of existing covariate values that fall in each marginal sampling strata.

Value

`optimDIST` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

`objDIST` returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

Author(s)

Alessandro Samuel-Rosa `<alessandrosamuelrosa@gmail.com>`

References

Hyndman, R. J.; Fan, Y. Sample quantiles in statistical packages. *The American Statistician*, v. 50, p. 361-365, 1996.

Everitt, B. S. *The Cambridge dictionary of statistics*. Cambridge: Cambridge University Press, p. 432, 2006.

See Also

`optimACDC`

Examples

```
require(sp)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
```

```
covars <- meuse.grid[, 5]
schedule <- scheduleSPSANN(initial.temperature = 1, chains = 1,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimDIST(points = 10, candi = candi, covars = covars,
                 use.coords = TRUE, schedule = schedule)
objSPSANN(res) -
  objDIST(points = res, candi = candi, covars = covars, use.coords = TRUE)
```

---

optimMKV *Optimization of sample configurations for spatial interpolation (II)*

Description

Optimize a sample configuration for spatial interpolation with a known linear model. A criterion is defined so that the sample configuration minimizes the mean or maximum kriging variance (**MKV**).

Usage

```
optimMKV(points, candi, covars, eqn = z ~ 1, vgm, krige.stat = "mean",
  ..., schedule = scheduleSPSANN(), plotit = FALSE, track = FALSE,
  boundary, progress = "txt", verbose = FALSE)

objMKV(points, candi, covars, eqn = z ~ 1, vgm, krige.stat = "mean",
  ...)
```

Arguments

points	Integer value, integer vector, data frame or matrix, or list. • Integer value. The number of points. These points will be randomly sampled from candi to form the starting sample configuration. • Integer vector. The row indexes of candi that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points. • Data frame or matrix. An object with three columns in the following order: [, "id"], the row indexes of candi that correspond to each point, [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates. • List. An object with two named sub-arguments: fixed, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and free, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.
candi	Data frame or matrix with the candidate locations for the jittered points. candi must have two columns in the following order: [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates.
covars	Data frame or matrix with the covariates in the columns.

eqn	Formula string that defines the dependent variable z as a linear model of the independent variables contained in covars. Defaults to eqn = z ~ 1, that is, ordinary kriging. See the argument formula in the function krige for more information.
vgm	Object of class variogramModel. See the argument model in the function krige for more information.
krige.stat	Character value defining the statistic that should be used to summarize the kriging variance. Available options are "mean" and "max" for the mean and maximum kriging variance, respectively. Defaults to krige.stat = "mean".
...	further arguments passed to krige.
schedule	List with 11 named sub-arguments defining the control parameters of the cooling schedule. See scheduleSPSANN.
plotit	(Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to plotit = FALSE.
track	(Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If track = FALSE (the default), only the starting and ending energy states are returned along with the results.
boundary	(Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and plotit = TRUE, boundary is estimated from candi.
progress	(Optional) Type of progress bar that should be used, with options "txt", for a text progress bar in the R console, "tk", to put up a Tk progress bar widget, and NULL to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to progress = "txt".
verbose	(Optional) Logical for printing messages about the progress of the optimization. Defaults to verbose = FALSE.

Details

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of spJitter.

Value

optimMKV returns an object of class OptimizedSampleConfiguration: the optimized sample configuration with details about the optimization.

objMKV returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

This function is based on the method originally proposed by Heuvelink, Brus and de Gruijter (2006) and implemented in the R-package **intamapInteractive** by Edzer Pebesma and Jon Skoien.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Brus, D. J.; Heuvelink, G. B. M. Optimization of sample patterns for universal kriging of environmental variables. *Geoderma*. v. 138, p. 86-95, 2007.

Heuvelink, G. B. M.; Brus, D. J.; de Gruijter, J. J. Optimization of sample configurations for digital mapping of soil properties with universal kriging. In: Lagacherie, P.; McBratney, A. & Voltz, M. (Eds.) *Digital soil mapping - an introductory perspective*. Elsevier, v. 31, p. 137-151, 2006.

Examples

```
## Not run:
data(meuse.grid, package = "sp")
candi <- meuse.grid[1:1000, 1:2]
covars <- as.data.frame(meuse.grid)[1:1000, ]
vgm <- gstat::vgm(psill = 10, model = "Exp", range = 500, nugget = 8)
schedule <- scheduleSPSANN(
  initial.temperature = 10, chains = 1, x.max = 1540, y.max = 2060,
  x.min = 0,  y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimMKV(
  points = 10, candi = candi, covars = covars, eqn = z ~ dist,
  vgm = vgm, schedule = schedule)
objSPSANN(res) - objMKV(
  points = res, candi = candi, covars = covars,  eqn = z ~ dist,
  vgm = vgm)

## End(Not run)
```

---

optimMSSD *Optimization of sample configurations for spatial interpolation (I)*

Description

Optimize a sample configuration for spatial interpolation. The criterion used is the mean squared shortest distance (**MSSD**) between sample points and prediction points.

Usage

```
optimMSSD(points, candi, schedule = scheduleSPSANN(), plotit = FALSE,
  track = FALSE, boundary, progress = "txt", verbose = FALSE)

objMSSD(points, candi)
```

Arguments

points	Integer value, integer vector, data frame or matrix, or list. • Integer value. The number of points. These points will be randomly sampled from candi to form the starting sample configuration. • Integer vector. The row indexes of candi that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points. • Data frame or matrix. An object with three columns in the following order: [, "id"], the row indexes of candi that correspond to each point, [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates. • List. An object with two named sub-arguments: fixed, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and free, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.
candi	Data frame or matrix with the candidate locations for the jittered points. candi must have two columns in the following order: [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates.
schedule	List with 11 named sub-arguments defining the control parameters of the cooling schedule. See scheduleSPSANN.
plotit	(Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to plotit = FALSE.
track	(Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If track = FALSE (the default), only the starting and ending energy states are returned along with the results.
boundary	(Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and plotit = TRUE, boundary is estimated from candi.
progress	(Optional) Type of progress bar that should be used, with options "txt", for a text progress bar in the R console, "tk", to put up a Tk progress bar widget, and NULL to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to progress = "txt".
verbose	(Optional) Logical for printing messages about the progress of the optimization. Defaults to verbose = FALSE.

Details

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of `spJitter`.

Spatial coverage sampling: Spatial coverage sampling is based on the knowledge that the kriging variance depends upon the distance between sample points. As such, the better the spread of the sample points in the spatial domain, the smaller the kriging variance. This is similar to using a regular grid of sample points. However, a regular grid usually is suboptimal for irregularly shaped areas.

Value

`optimMSSD` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

`objMSSD` returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

This function was derived with modifications from the method known as *spatial coverage sampling* originally proposed by Brus, de Gruijter and van Groenigen (2006), and implemented in the R-package **spcosa** by Dennis Walvoort, Dick Brus and Jaap de Gruijter.

Author(s)

Alessandro Samuel-Rosa `<alessandrosamuelrosa@gmail.com>`

References

Brus, D. J.; de Gruijter, J. J.; van Groenigen, J.-W. Designing spatial coverage samples using the k-means clustering algorithm. In: P. Lagacherie,A. M.; Voltz, M. (Eds.) *Digital soil mapping – an introductory perspective*. Elsevier, v. 31, p. 183-192, 2006.

de Gruijter, J. J.; Brus, D.; Bierkens, M.; Knotters, M. *Sampling for natural resource monitoring*. Berlin: Springer, p. 332, 2006.

Walvoort, D. J. J.; Brus, D. J.; de Gruijter, J. J. An R package for spatial coverage sampling and random sampling from compact geographical strata by k-means. *Computers and Geosciences*. v. 36, p. 1261-1267, 2010.

See Also

`[distanceFromPoints](https://CRAN.R-project.org/package=raster)`, `[stratify](https://CRAN.R-project.o`

Examples

```
require(sp)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
schedule <- scheduleSPSANN(chains = 1, initial.temperature = 5000000,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimMSSD(points = 10, candi = candi, schedule = schedule)
objSPSANN(res) - objMSSD(candi = candi, points = res)
```

---

optimPPL *Optimization of sample configurations for variogram identification and estimation*

Description

Optimize a sample configuration for variogram identification and estimation. A criterion is defined so that the optimized sample configuration has a given number of points or point-pairs contributing to each lag-distance class (**PPL**).

Usage

```
optimPPL(points, candi, lags = 7, lags.type = "exponential",
  lags.base = 2, cutoff, criterion = "distribution", distri,
  pairs = FALSE, schedule = scheduleSPSANN(), plotit = FALSE,
  track = FALSE, boundary, progress = "txt", verbose = FALSE)

objPPL(points, candi, lags = 7, lags.type = "exponential",
  lags.base = 2, cutoff, distri, criterion = "distribution",
  pairs = FALSE, x.max, x.min, y.max, y.min)

countPPL(points, candi, lags = 7, lags.type = "exponential",
  lags.base = 2, cutoff, pairs = FALSE, x.max, x.min, y.max, y.min)
```

Arguments

`points` Integer value, integer vector, data frame or matrix, or list.

- Integer value. The number of points. These points will be randomly sampled from `candi` to form the starting sample configuration.
- Integer vector. The row indexes of `candi` that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points.
- Data frame or matrix. An object with three columns in the following order: `[, "id"]`, the row indexes of `candi` that correspond to each point, `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.

- List. An object with two named sub-arguments: `fixed`, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and `free`, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.

`candi` Data frame or matrix with the candidate locations for the jittered points. `candi` must have two columns in the following order: `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.

`lags` Integer value, the number of lag-distance classes. Alternatively, a vector of numeric values with the lower and upper bounds of each lag-distance class, the lowest value being larger than zero (e.g. 0.0001). Defaults to `lags = 7`.

`lags.type` Character value, the type of lag-distance classes, with options `"equidistant"` and `"exponential"`. Defaults to `lags.type = "exponential"`.

`lags.base` Numeric value, base of the exponential expression used to create exponentially spaced lag-distance classes. Used only when `lags.type = "exponential"`. Defaults to `lags.base = 2`.

`cutoff` Numeric value, the maximum distance up to which lag-distance classes are created. Used only when `lags` is an integer value. If missing, it is set to be equal to the length of the diagonal of the rectangle with sides `x.max` and `y.max` as defined in `scheduleSPSANN`.

`criterion` Character value, the feature used to describe the energy state of the system configuration, with options `"minimum"` and `"distribution"`. Defaults to `objective = "distribution"`.

`distri` Numeric vector, the distribution of points or point-pairs per lag-distance class that should be attained at the end of the optimization. Used only when `criterion = "distribution"`. Defaults to a uniform distribution.

`pairs` Logical value. Should the sample configuration be optimized regarding the number of point-pairs per lag-distance class? Defaults to `pairs = FALSE`.

`schedule` List with 11 named sub-arguments defining the control parameters of the cooling schedule. See `scheduleSPSANN`.

`plotit` (Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to `plotit = FALSE`.

`track` (Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If `track = FALSE` (the default), only the starting and ending energy states are returned along with the results.

`boundary` (Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and `plotit = TRUE`, `boundary` is estimated from `candi`.

`progress` (Optional) Type of progress bar that should be used, with options `"txt"`, for a text progress bar in the R console, `"tk"`, to put up a Tk progress bar widget, and `NULL` to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to `progress = "txt"`.

`verbose` (Optional) Logical for printing messages about the progress of the optimization. Defaults to `verbose = FALSE`.

`x.max` Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The minimum quantity should be equal to, at least, the minimum distance between two neighbouring candidate locations. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from `candi`.

`x.min` Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The minimum quantity should be equal to, at least, the minimum distance between two neighbouring candidate locations. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from `candi`.

`y.max` Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The minimum quantity should be equal to, at least, the minimum distance between two neighbouring candidate locations. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from `candi`.

`y.min` Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The minimum quantity should be equal to, at least, the minimum distance between two neighbouring candidate locations. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from `candi`.

Details

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of `spJitter`.

Lag-distance classes: Two types of lag-distance classes can be created by default. The first are evenly spaced lags (`lags.type = "equidistant"`). They are created by simply dividing the distance interval from 0.0001 to `cutoff` by the required number of lags. The minimum value of 0.0001 guarantees that a point does not form a pair with itself. The second type of lags is defined by exponential spacings (`lags.type = "exponential"`). The spacings are defined by the base b of the exponential expression b^n, where n is the required number of lags. The base is defined using the argument `lags.base`. See `vgmLags` for other details.

Using the default uniform distribution means that the number of point-pairs per lag-distance class (`pairs = TRUE`) is equal to $n \times (n-1)/(2 \times lag)$, where n is the total number of points and lag is the number of lags. If `pairs = FALSE`, then it means that the number of points per lag is equal to the total number of points. This is the same as expecting that each point contributes to every lag. Distributions other than the available options can be easily implemented changing the arguments `lags` and `distri`.

There are two optimizing criteria implemented. The first is called using `criterion = "distribution"` and is used to minimize the sum of the absolute differences between a pre-specified distribution and the observed distribution of points or point-pairs per lag-distance class. The second criterion is called using `criterion = "minimum"`. It corresponds to maximizing the minimum number of points or point-pairs observed over all lag-distance classes.

Value

`optimPPL` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

`objPPL` returns a numeric value: the energy state of the sample configuration – the objective function value.

`countPPL` returns a data.frame with three columns: a) the lower and b) upper limits of each lag-distance class, and c) the number of points or point-pairs per lag-distance class.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Bresler, E.; Green, R. E. *Soil parameters and sampling scheme for characterizing soil hydraulic properties of a watershed*. Honolulu: University of Hawaii at Manoa, p. 42, 1982.

Pettitt, A. N.; McBratney, A. B. Sampling designs for estimating spatial variance components. *Applied Statistics*. v. 42, p. 185, 1993.

Russo, D. Design of an optimal sampling network for estimating the variogram. *Soil Science Society of America Journal*. v. 48, p. 708-716, 1984.

Truong, P. N.; Heuvelink, G. B. M.; Gosling, J. P. Web-based tool for expert elicitation of the variogram. *Computers and Geosciences*. v. 51, p. 390-399, 2013.

Warrick, A. W.; Myers, D. E. Optimization of sampling locations for variogram calculations. *Water Resources Research*. v. 23, p. 496-500, 1987.

Examples

```
## Not run:
# This example takes more than 5 seconds
require(sp)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
schedule <- scheduleSPSANN(chains = 1, initial.temperature = 30,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimPPL(points = 10, candi = candi, schedule = schedule)
objSPSANN(res) - objPPL(points = res, candi = candi)
countPPL(points = res, candi = candi)
```

```
## End(Not run)
```

---

optimSPAN *Optimization of sample configurations for variogram and spatial trend identification and estimation, and for spatial interpolation*

Description

Optimize a sample configuration for variogram and spatial trend identification and estimation, and for spatial interpolation. An utility function U is defined so that the sample points cover, extend over, spread over, **SPAN** the feature, variogram and geographic spaces. The utility function is obtained aggregating four objective functions: **CORR**, **DIST**, **PPL**, and **MSSD**.

Usage

```
optimSPAN(points, candi, covars, strata.type = "area",
  use.coords = FALSE, lags = 7, lags.type = "exponential",
  lags.base = 2, cutoff, criterion = "distribution", distri,
  pairs = FALSE, schedule = scheduleSPSANN(), plotit = FALSE,
  track = FALSE, boundary, progress = "txt", verbose = FALSE,
  weights, nadir = list(sim = NULL, seeds = NULL, user = NULL, abs =
  NULL), utopia = list(user = NULL, abs = NULL))

objSPAN(points, candi, covars, strata.type = "area",
  use.coords = FALSE, lags = 7, lags.type = "exponential",
  lags.base = 2, cutoff, criterion = "distribution", distri,
  pairs = FALSE, x.max, x.min, y.max, y.min, weights, nadir = list(sim
  = NULL, seeds = NULL, user = NULL, abs = NULL), utopia = list(user =
  NULL, abs = NULL))
```

Arguments

`points` Integer value, integer vector, data frame or matrix, or list.

- Integer value. The number of points. These points will be randomly sampled from `candi` to form the starting sample configuration.
- Integer vector. The row indexes of `candi` that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points.
- Data frame or matrix. An object with three columns in the following order: `[, "id"]`, the row indexes of `candi` that correspond to each point, `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.
- List. An object with two named sub-arguments: `fixed`, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and `free`, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.

`candi` Data frame or matrix with the candidate locations for the jittered points. `candi` must have two columns in the following order: `[, "x"]`, the projected x-coordinates, and `[, "y"]`, the projected y-coordinates.

`covars` Data frame or matrix with the covariates in the columns.

`strata.type` (Optional) Character value setting the type of stratification that should be used to create the marginal sampling strata (or factor levels) for the numeric covariates. Available options are `"area"`, for equal-area, and `"range"`, for equal-range. Defaults to `strata.type = "area"`.

`use.coords` (Optional) Logical value. Should the spatial x- and y-coordinates be used as covariates? Defaults to `use.coords = FALSE`.

`lags` Integer value, the number of lag-distance classes. Alternatively, a vector of numeric values with the lower and upper bounds of each lag-distance class, the lowest value being larger than zero (e.g. 0.0001). Defaults to `lags = 7`.

`lags.type` Character value, the type of lag-distance classes, with options `"equidistant"` and `"exponential"`. Defaults to `lags.type = "exponential"`.

`lags.base` Numeric value, base of the exponential expression used to create exponentially spaced lag-distance classes. Used only when `lags.type = "exponential"`. Defaults to `lags.base = 2`.

`cutoff` Numeric value, the maximum distance up to which lag-distance classes are created. Used only when `lags` is an integer value. If missing, it is set to be equal to the length of the diagonal of the rectangle with sides `x.max` and `y.max` as defined in `scheduleSPSANN`.

`criterion` Character value, the feature used to describe the energy state of the system configuration, with options `"minimum"` and `"distribution"`. Defaults to `objective = "distribution"`.

`distri` Numeric vector, the distribution of points or point-pairs per lag-distance class that should be attained at the end of the optimization. Used only when `criterion = "distribution"`. Defaults to a uniform distribution.

`pairs` Logical value. Should the sample configuration be optimized regarding the number of point-pairs per lag-distance class? Defaults to `pairs = FALSE`.

`schedule` List with 11 named sub-arguments defining the control parameters of the cooling schedule. See `scheduleSPSANN`.

`plotit` (Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to `plotit = FALSE`.

`track` (Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If `track = FALSE` (the default), only the starting and ending energy states are returned along with the results.

`boundary` (Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and `plotit = TRUE`, `boundary` is estimated from `candi`.

`progress` (Optional) Type of progress bar that should be used, with options `"txt"`, for a text progress bar in the R console, `"tk"`, to put up a Tk progress bar widget, and

NULL to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to progress = "txt".

verbose — (Optional) Logical for printing messages about the progress of the optimization. Defaults to verbose = FALSE.

weights — List with named sub-arguments. The weights assigned to each one of the objective functions that form the multi-objective combinatorial optimization problem. They must be named after the respective objective function to which they apply. The weights must be equal to or larger than 0 and sum to 1.

nadir — List with named sub-arguments. Three options are available: 1) sim – the number of simulations that should be used to estimate the nadir point, and seeds – vector defining the random seeds for each simulation; 2) user – a list of user-defined nadir values named after the respective objective functions to which they apply; 3) abs – logical for calculating the nadir point internally (experimental).

utopia — List with named sub-arguments. Two options are available: 1) user – a list of user-defined values named after the respective objective functions to which they apply; 2) abs – logical for calculating the utopia point internally (experimental).

x.max, x.min, y.max, y.min — Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The minimum quantity should be equal to, at least, the minimum distance between two neighbouring candidate locations. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from candi.

Details

The help page of minmaxPareto contains details on how **spsann** solves the multi-objective combinatorial optimization problem of finding a globally optimum sample configuration that meets multiple, possibly conflicting, sampling objectives.

Details about the mechanism used to generate a new sample configuration out of the current sample configuration by randomly perturbing the coordinates of a sample point are available in the help page of spJitter.

Visit the help pages of optimCORR, optimDIST, optimPPL, and optimMSSD to see the details of the objective functions that compose **SPAN**.

Value

optimSPAN returns an object of class OptimizedSampleConfiguration: the optimized sample configuration with details about the optimization.

objSPAN returns a numeric value: the energy state of the sample configuration – the objective function value.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

See Also

optimCORR, optimDIST, optimPPL, optimMSSD

Examples

```
## Not run:
# This example takes more than 5 seconds to run!
require(sp)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
nadir <- list(sim = 10, seeds = 1:10)
utopia <- list(user = list(DIST = 0, CORR = 0, PPL = 0, MSSD = 0))
covars <- meuse.grid[, 5]
schedule <- scheduleSPSANN(chains = 1, initial.temperature = 1,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)
weights <- list(CORR = 1/6, DIST = 1/6, PPL = 1/3, MSSD = 1/3)
set.seed(2001)
res <- optimSPAN(
  points = 10, candi = candi, covars = covars, nadir = nadir, weights = weights,
    use.coords = TRUE, utopia = utopia, schedule = schedule)
objSPSANN(res) -
  objSPAN(points = res, candi = candi, covars = covars, nadir = nadir,
            use.coords = TRUE, utopia = utopia, weights = weights)

## End(Not run)
```

---

optimUSER — *Optimization of sample configurations using a user-defined objective function*

Description

Optimize a sample configuration using a user-defined objective function.

Usage

```
optimUSER(points, candi, fun, ..., schedule = scheduleSPSANN(),
  plotit = FALSE, track = FALSE, boundary, progress = "txt",
  verbose = FALSE)
```

Arguments

points	Integer value, integer vector, data frame or matrix, or list. • Integer value. The number of points. These points will be randomly sampled from candi to form the starting sample configuration. • Integer vector. The row indexes of candi that correspond to the points that form the starting sample configuration. The length of the vector defines the number of points. • Data frame or matrix. An object with three columns in the following order: [, "id"], the row indexes of candi that correspond to each point, [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates. • List. An object with two named sub-arguments: fixed, a data frame or matrix with the projected x- and y-coordinates of the existing sample configuration – kept fixed during the optimization –, and free, an integer value defining the number of points that should be added to the existing sample configuration – free to move during the optimization.
candi	Data frame or matrix with the candidate locations for the jittered points. candi must have two columns in the following order: [, "x"], the projected x-coordinates, and [, "y"], the projected y-coordinates.
fun	A function defining the objective function that should be used to evaluate the energy state of the system configuration at each random perturbation of a candidate sample point. See 'Details' for more information.
...	Other arguments passed to the objective function. See 'Details' for more information.
schedule	List with 11 named sub-arguments defining the control parameters of the cooling schedule. See scheduleSPSANN.
plotit	(Optional) Logical for plotting the optimization results, including a) the progress of the objective function, and b) the starting (gray circles) and current sample configuration (black dots), and the maximum jitter in the x- and y-coordinates. The plots are updated at each 10 jitters. When adding points to an existing sample configuration, fixed points are indicated using black crosses. Defaults to plotit = FALSE.
track	(Optional) Logical value. Should the evolution of the energy state be recorded and returned along with the result? If track = FALSE (the default), only the starting and ending energy states are returned along with the results.
boundary	(Optional) SpatialPolygon defining the boundary of the spatial domain. If missing and plotit = TRUE, boundary is estimated from candi.
progress	(Optional) Type of progress bar that should be used, with options "txt", for a text progress bar in the R console, "tk", to put up a Tk progress bar widget, and NULL to omit the progress bar. A Tk progress bar widget is useful when using parallel processors. Defaults to progress = "txt".
verbose	(Optional) Logical for printing messages about the progress of the optimization. Defaults to verbose = FALSE.

Details

The user-defined objective function `fun` must be an object of class `function` and include the argument `points`. The argument `points` is defined in `optimUSER` as a matrix with three columns: `[, 1]` the identification of each sample point given by the respective row indexes of `candi`, `[, 2]` the x-coordinates, and `[, 3]` the y-coordinates. The identification is useful to retrieve information from any data matrix used by the objective function defined by the user.

Value

`optimUSER` returns an object of class `OptimizedSampleConfiguration`: the optimized sample configuration with details about the optimization.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

Examples

```
## Not run:
# This example takes more than 5 seconds
require(sp)
require(SpatialTools)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
schedule <- scheduleSPSANN(chains = 1, initial.temperature = 30,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)

# Define the objective function - number of points per lag distance class
objUSER <-
  function (points, lags, n_lags, n_pts) {
    dm <- SpatialTools::dist1(points[, 2:3])
    ppl <- vector()
    for (i in 1:n_lags) {
      n <- which(dm > lags[i] & dm <= lags[i + 1], arr.ind = TRUE)
      ppl[i] <- length(unique(c(n)))
    }
    distri <- rep(n_pts, n_lags)
    res <- sum(distri - ppl)
  }
lags <- seq(1, 1000, length.out = 10)

# Run the optimization using the user-defined objective function
set.seed(2001)
timeUSER <- Sys.time()
resUSER <- optimUSER(points = 10, fun = objUSER, lags = lags, n_lags = 9,
                     n_pts = 10, candi = candi, schedule = schedule)
timeUSER <- Sys.time() - timeUSER

# Run the optimization using the respective function implemented in spsann
set.seed(2001)
```

```
timePPL <- Sys.time()
resPPL <- optimPPL(points = 10, candi = candi, lags = lags,
                   schedule = schedule)
timePPL <- Sys.time() - timePPL

# Compare results
timeUSER
timePPL
lapply(list(resUSER, resPPL), countPPL, candi = candi, lags = lags)
objSPSANN(resUSER) - objSPSANN(resPPL)

## End(Not run)
```

`plot.OptimizedSampleConfiguration`
Plot an optimized sample configuration

Description

Plot the evolution of the energy state and the optimized sample configuration

Usage

```
## S3 method for class 'OptimizedSampleConfiguration'
plot(x, which = 1:2, boundary,
  ...)
```

Arguments

`x`	Object of class `OptimizedSampleConfiguration` returned by one of the `optim`-functions.
`which`	Which plot should be produced: evolution of the energy state (1), optimized sample configuration (2), or both (1:2)? Defaults to `which = 1:2`.
`boundary`	Object of class `Spatial` defining the boundary of the sampling region.
`...`	Other options passed to `plot`.

Examples

```
require(sp)
data(meuse.grid)
candi <- meuse.grid[, 1:2]
covars <- meuse.grid[, 5]
schedule <- scheduleSPSANN(initial.temperature = 5, chains = 1,
                           x.max = 1540, y.max = 2060, x.min = 0,
                           y.min = 0, cellsize = 40)
set.seed(2001)
res <- optimCORR(points = 10, candi = candi, covars = covars,
                 use.coords = TRUE, schedule = schedule)
plot(res)
```

scheduleSPSANN **spsann** *annealing schedule*

Description

Set the control parameters for the annealing schedule of **spsann** functions.

Usage

```
scheduleSPSANN(initial.acceptance = 0.95, initial.temperature = 0.001,
  temperature.decrease = 0.95, chains = 500, chain.length = 1,
  stopping = 10, x.max, x.min = 0, y.max, y.min = 0, cellsize)
```

Arguments

`initial.acceptance`
: Numeric value between 0 and 1 defining the initial acceptance probability, i.e. the proportion of proposed system configurations that should be accepted in the first chain. The optimization is stopped and a warning is issued if this value is not attained. Defaults to `initial.acceptance = 0.95`.

`initial.temperature`
: Numeric value larger than 0 defining the initial temperature of the system. A low `initial.temperature`, combined with a low `initial.acceptance` result in the algorithm to behave as a greedy algorithm, i.e. only better system configurations are accepted. Defaults to `initial.temperature = 0.001`.

`temperature.decrease`
: Numeric value between 0 and 1 used as a multiplying factor to decrease the temperature at the end of each Markov chain. Defaults to `temperature.decrease = 0.95`.

`chains`
: Integer value defining the maximum number of chains, i.e. the number of cycles of jitters at which the temperature and the size of the neighbourhood should be kept constant. Defaults to `chains = 500`.

`chain.length`
: Integer value defining the length of each Markov chain relative to the number of sample points. Defaults to `chain.length = 1`, i.e. one time the number of sample points.

`stopping`
: Integer value defining the maximum allowable number of Markov chains without improvement of the objective function value. Defaults to `stopping = 10`.

`x.max, x.min, y.max, y.min`
: Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from `candi`, `x.min` and `y.min` being set to zero, and `x.max` and `y.max` being set to half the maximum distance in the x- and y-coordinates, respectively.

`cellsize`
: Vector with two numeric values defining the horizontal (x) and vertical (y) spacing between the candidate locations in `candi`. A single value can be used if the spacing in the x- and y-coordinates is the same. If `cellsize = 0` then **spsann** understands that a finite set of candidate locations is being used (See Details).

Value

A list with a set of control parameters of the annealing schedule.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Aarts, E. H. L.; Korst, J. H. M. Boltzmann machines for travelling salesman problems. *European Journal of Operational Research*, v. 39, p. 79-95, 1989.

Černý, V. Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, v. 45, p. 41-51, 1985.

Brus, D. J.; Heuvelink, G. B. M. Optimization of sample patterns for universal kriging of environmental variables. *Geoderma*, v. 138, p. 86-95, 2007.

Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. Optimization by simulated annealing. *Science*, v. 220, p. 671-680, 1983.

Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, v. 21, p. 1087-1092, 1953.

van Groenigen, J.-W.; Stein, A. Constrained optimization of spatial sampling using continuous simulated annealing. *Journal of Environmental Quality*. v. 27, p. 1078-1086, 1998.

Webster, R.; Lark, R. M. *Field sampling for environmental science and management*. London: Routledge, p. 200, 2013.

See Also

optimACDC, optimCORR, optimDIST, optimMKV, optimMSSD, optimPPL, optimSPAN, optimUSER.

Examples

```
schedule <- scheduleSPSANN()
```

---

spJitter *Random perturbation of spatial points*

Description

Randomly perturb ('jitter') the coordinates of spatial points.

Usage

```
spJitter(points, candi, x.max, x.min, y.max, y.min, which.point, cellsize)
```

Arguments

points	Data frame or matrix with three columns in the following order: [, "id"] the row indexes of candi that correspond to each point, [, "x"] the projected x-coordinates, and [, "y"] the projected y-coordinates. Note that points must be a subset of candi.
candi	Data frame or matrix with the candidate locations for the jittered points. candi must have two columns in the following order: [, "x"] the projected x-coordinates, and [, "y"] the projected y-coordinates.
x.max, x.min, y.max, y.min	Numeric value defining the minimum and maximum quantity of random noise to be added to the projected x- and y-coordinates. The minimum quantity should be equal to, at least, the minimum distance between two neighbouring candidate locations. The units are the same as of the projected x- and y-coordinates. If missing, they are estimated from candi.
which.point	Integer values defining which point should be perturbed.
cellsize	Vector with two numeric values defining the horizontal (x) and vertical (y) spacing between the candidate locations in candi. A single value can be used if the spacing in the x- and y-coordinates is the same. If cellsize = 0 then **spsann** understands that a finite set of candidate locations is being used (See Details).

Details

Jittering methods: There are multiple mechanism to generate a new sample configuration out of the current sample configuration. The main step consists of randomly perturbing the coordinates of a sample point, a process known as 'jittering'. These mechanisms can be classified based on how the set of candidate locations is defined. For example, one could use an *infinite* set of candidate locations, that is, any location in the sampling region can be selected as the new location of a jittered point. All that is needed is a polygon indicating the boundary of the sampling region. This method is the most computationally demanding because every time a point is jittered, it is necessary to check if the point falls in sampling region.

Another approach consists of using a *finite* set of candidate locations for the jittered points. A finite set of candidate locations is created by discretising the sampling region, that is, creating a fine grid of points that serve as candidate locations for the jittered point. This is the least computationally demanding jittering method because, by definition, the jittered point will always fall in the sampling region.

Using a finite set of candidate locations has two important inconveniences. First, not all locations in the sampling region can be selected as the new location for a jittered point. Second, when a point is jittered, it may be that the new location already is occupied by another point. If this happens, another location has to be iteratively sought for, say, as many times as the number of points in the sample. In general, the more points there are in the sample, the more likely it is that the new location already is occupied by another point. If a solution is not found in a reasonable time, the point selected to be jittered is kept in its original location. Such a procedure clearly is suboptimal.

spsann uses a more elegant method which is based on using a finite set of candidate locations coupled with a form of *two-stage random sampling* as implemented in [spsample](https://CRAN.R-project.org/package
Because the candidate locations are placed on a finite regular grid, they can be seen as the centre nodes of a finite set of grid cells (or pixels of a raster image). In the first stage, one of the "grid

cells" is selected with replacement, i.e. independently of already being occupied by another sample point. The new location for the point chosen to be jittered is selected within that "grid cell" by simple random sampling. This method guarantees that virtually any location in the sampling region can be selected. It also discards the need to check if the new location already is occupied by another point, speeding up the computations when compared to the first two approaches.

Value

A matrix with the jittered projected coordinates of the points.

Note

The distance between two points is computed as the Euclidean distance between them. This computation assumes that the optimization is operating in the two-dimensional Euclidean space, i.e. the coordinates of the sample points and candidate locations should not be provided as latitude/longitude. **spsann** has no mechanism to check if the coordinates are projected: the user is responsible for making sure that this requirement is attained.

Author(s)

Alessandro Samuel-Rosa <alessandrosamuelrosa@gmail.com>

References

Edzer Pebesma, Jon Skoien with contributions from Olivier Baume, A. Chorti, D.T. Hristopulos, S.J. Melles and G. Spiliopoulos (2013). *intamapInteractive: procedures for automated interpolation - methods only to be used interactively, not included in* `intamap` *package.* R package version 1.1-10.

van Groenigen, J.-W. *Constrained optimization of spatial sampling: a geostatistical approach.* Wageningen: Wageningen University, p. 148, 1999.

Walvoort, D. J. J.; Brus, D. J.; de Gruijter, J. J. An R package for spatial coverage sampling and random sampling from compact geographical strata by k-means. *Computers & Geosciences*. v. 36, p. 1261-1267, 2010.

See Also

`ssaOptim`, `zerodist`, `jitter`, `[jitter2d](https://CRAN.R-project.org/package=geoR)`.

Examples

```
require(sp)
data(meuse.grid)
meuse.grid <- as.matrix(meuse.grid[, 1:2])
meuse.grid <- matrix(cbind(1:dim(meuse.grid)[1], meuse.grid), ncol = 3)
pts1 <- sample(c(1:dim(meuse.grid)[1]), 155)
pts2 <- meuse.grid[pts1, ]
pts3 <- spJitter(points = pts2, candi = meuse.grid, x.min = 40,
                 x.max = 100, y.min = 40, y.max = 100,
                 which.point = 10, cellsize = 40)
plot(meuse.grid[, 2:3], asp = 1, pch = 15, col = "gray")
```

```
points(pts2[, 2:3], col = "red", cex = 0.5)
points(pts3[, 2:3], pch = 19, col = "blue", cex = 0.5)

#' Cluster of points
pts1 <- c(1:55)
pts2 <- meuse.grid[pts1, ]
pts3 <- spJitter(points = pts2, candi = meuse.grid, x.min = 40,
                 x.max = 80, y.min = 40, y.max = 80,
                 which.point = 1, cellsize = 40)
plot(meuse.grid[, 2:3], asp = 1, pch = 15, col = "gray")
points(pts2[, 2:3], col = "red", cex = 0.5)
points(pts3[, 2:3], pch = 19, col = "blue", cex = 0.5)
```

Index