spsann

Optimization of Sample Configurations Using Spatial Simulated Annealing

Alessandro Samuel-Rosa*

April 29, 2019

Contents

1 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

*Departamento de Solos, Universidade Federal Rural do Rio de Janeiro, Seropédica, Rio de Janeiro, Brasil. alessandrosamuelrosa@gmail.com

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.

2 Getting Started

2.1 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.

2.2 Planning our R script

Now that we are acquainted with the structure and naming conventions of **spsann**, it is time start planning how to prepare and organize our R script that will be used to optimize a sample configuration. Overall, there are four steps that need to be followed, i.e. our R script will contain four main sections (see below).

The first thing to do after we decide upon the objective function that will be used to optimize our sample configuration is to load and pre-process all the data required by the chosen objective function. This step can be time consuming depending on the complexity of the optimization problem, on the quality of the existing data for the problem at hand, and on the requirements of the chosen objective function.

```
# OPTIMIZING A SAMPLE CONFIGURATION IN FOUR STEPS
# Step 1. Load and pre-process the data
# ...
# Step 2. Set control parameters
# ...
# Step 3. Execute the simulated annealing algorithm
# ...
# ... Be prepared to wait!!!
# ...
# Step 4. Evaluate the optimized sample configuration
# ...
```

The next step consists of setting the parameters that control the spatial simulated annealing algorithm. This usually requires an element of trial-and-error. As such, it will likely take longer for a less experienced user to set the appropriate control parameters.

The third step consists of executing the spatial simulated annealing algorithm with the chosen objective function. Depending on the complexity of the optimization problem and on the processing capacity of the computer, the execution can last from minutes to several hours or even days. Thus, we strongly advise the user to first evaluate if the installed processing capacity is enough to solve the optimization problem within a reasonable time.

Finally, with the optimized sample configuration at hand, we need to evaluate if it meets all our requirements. This can be done using the information returned by the spatial simulated annealing algorithm, plotting the optimized sample configuration, or any other means that the user might find useful.

3 Optimizing Sample Configurations

3.1 Spatial trend estimation

Our first example is about the optimization of sample configurations for spatial trend identification and estimation. A objective function is defined so that the sample reproduces the marginal distribution of the covariates.

```
# Load and pre-process the data
data(meuse.grid, package = "sp")
boundary <- meuse.grid
sp::coordinates(boundary) <- c("x", "y")
sp::gridded(boundary) <- TRUE
boundary <- rgeos::gUnaryUnion(as(boundary, "SpatialPolygons"))
candi <- meuse.grid[, 1:2]
covars <- meuse.grid[, 6:7]

# Set control parameters
schedule <- scheduleSPSANN(initial.temperature = 0.5)
set.seed(2001)

# Execute the simulated annealing algorithm
res <- optimDIST(
  points = 30, candi = candi, covars = covars, use.coords = TRUE,
  schedule = schedule, plotit = TRUE, boundary = boundary)

# Evaluate the optimized sample configuration
objSPSANN(res)
objDIST(
  points = res, candi = candi, covars = covars, use.coords = TRUE)
```

```r
plot(res, boundary = boundary)
```

3.2 Variogram estimation

The second example shows the optimization of sample configurations for variogram identification and estimation using an objective function proposed in the 1980s. It consists of aiming at a uniform distribution of point-pairs per lag-distance class.

```r
# Load and pre-process the data
data(meuse.grid, package = "sp")
boundary <- meuse.grid
sp::coordinates(boundary) <- c("x", "y")
sp::gridded(boundary) <- TRUE
boundary <- rgeos::gUnaryUnion(as(boundary, "SpatialPolygons"))
candi <- meuse.grid[, 1:2]

# Set control parameters
schedule <- scheduleSPSANN(initial.temperature = 500)
set.seed(2001)

# Execute the simulated annealing algorithm
res <- optimPPL(
  points = 30, candi = candi, pairs = TRUE, schedule = schedule,
  plotit = TRUE, boundary = boundary)

# Evaluate the optimized sample configuration
objSPSANN(res)
objPPL(points = res, pairs = TRUE, candi = candi)
countPPL(points = res, candi = candi, pairs = TRUE)
plot(res, boundary = boundary)
```

3.3 Spatial interpolation

For the sake of spatial interpolation, the third example with explore an objective function that aims at minimizing the mean squared shortest distance between sample points and prediction points. The difference here is that the optimization is performed using a greedy algorithm.

```r
# Load and pre-process the data
data(meuse.grid, package = "sp")
boundary <- meuse.grid
sp::coordinates(boundary) <- c("x", "y")
sp::gridded(boundary) <- TRUE
```

```
boundary <- rgeos::gUnaryUnion(as(boundary, "SpatialPolygons"))
candi <- meuse.grid[, 1:2]

# Set control parameters
schedule <- scheduleSPSANN(
  initial.acceptance = 0, initial.temperature = 0.01)
set.seed(2001)

# Execute the simulated annealing algorithm
res <- optimMSSD(
  points = 30, candi = candi, schedule = schedule, plotit = TRUE,
  boundary = boundary)

# Evaluate the optimized sample configuration
objSPSANN(res)
objMSSD(candi = candi, points = res)
plot(res, boundary = boundary)
```

A different objective function can be used to optimize our sample configuration if we know the form of the model of spatial variation that will be used for spatial interpolation. In this case the goal could be to minimize the mean universal kriging variance.

```
# Load and pre-process the data
data(meuse.grid, package = "sp")
boundary <- meuse.grid
sp::coordinates(boundary) <- c("x", "y")
sp::gridded(boundary) <- TRUE
boundary <- rgeos::gUnaryUnion(as(boundary, "SpatialPolygons"))
candi <- meuse.grid[, 1:2]
covars <- as.data.frame(meuse.grid)

# Set control parameters
vgm <- gstat::vgm(
  psill = 10, model = "Exp", range = 500, nugget = 8)
schedule <- scheduleSPSANN(initial.temperature = 10)
set.seed(2001)

# Execute the simulated annealing algorithm
res <- optimMKV(
  points = 30, candi = candi, covars = covars, vgm = vgm,
  eqn = z ~ soil, plotit = TRUE, boundary = boundary,
  schedule = schedule)

# Evaluate the optimized sample configuration
```

```
objSPSANN(res)
objMKV(
  points = res, candi = candi, covars = covars,
  eqn = z ~ soil, vgm = vgm)
plot(res, boundary = boundary)
```

3.4 User defined objective function

The user can define her/his own objective function and plug it in a general purpose function implemented in **spsann**. Let us see an example when an objective function is defined so that the sample will have all points forming point-pairs at all lag-distance classes.

```
# Load and pre-process the data
data(meuse.grid, package = "sp")
boundary <- meuse.grid
sp::coordinates(boundary) <- c("x", "y")
sp::gridded(boundary) <- TRUE
boundary <- rgeos::gUnaryUnion(as(boundary, "SpatialPolygons"))
candi <- meuse.grid[, 1:2]

# Set control parameters
schedule <- scheduleSPSANN(
  initial.temperature = 30, x.max = 1540, y.max = 2060,
  x.min = 0, y.min = 0, cellsize = 40)
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)
set.seed(2001)

# Execute the simulated annealing algorithm
res <- optimUSER(
  points = 30, fun = objUSER, lags = lags, n_lags = 9,
  n_pts = 10, candi = candi, schedule = schedule,
  plotit = TRUE, boundary = boundary)
```

```r
# Evaluate the optimized sample configuration
objSPSANN(res)
countPPL(res, candi = candi, lags = lags)
plot(res, boundary = boundary)
```

```r
# library(magrittr)
#
# # Load existing sampling grid
# data(meuse, package = "sp")
# meuse <- sf::st_as_sf(meuse, coords = c('x', 'y'))
#
# # Set control parameters
# bb <- sf::st_bbox(meuse)
# x.max <- diff(bb[c('xmin', 'xmax')])
# y.max <- diff(bb[c('ymin', 'ymax')])
# schedule <-
#   spsann::scheduleSPSANN(
#     initial.temperature = 100000, chains = 500,
#     # x.max = x.max, y.max = y.max, x.min = 0, y.min = 0,
#     cellsize = 0)
#
# # Execute the simulated annealing algorithm
# n <- round(nrow(meuse) / 4)
# res <- spsann::optimMSSD(
#   points = n,
#   candi = sf::st_coordinates(meuse) %>% `colnames<-`(c('x', 'y')),
#   schedule = schedule, plotit = TRUE)
```