Package 'recmap'

June 3, 2019

Type Package

Title Compute the Rectangular Statistical Cartogram

Version 1.0.5

Maintainer Christian Panse <Christian.Panse@gmail.com>

Description Provides an interface and a C++ implementation of the RecMap MP2 construction heuristic (Panse C. (2018) <doi:10.18637/jss.v086.c01>). This algorithm draws maps according to a given statistical value (e.g., election results, population or epidemiological data). The basic idea of the RecMap algorithm is that each map region (e.g., different countries) is represented by a rectangle. The area of each rectangle represents the statistical value given as input (maintain zero cartographic error). Documentation about the usage of the recmap algorithm is provided by a vignette included in this package.

License GPL-3

Depends R (>= 3.5), GA (>= 3.1), Rcpp (>= 1.0), sp(>= 1.3)

Suggests colorspace, doParallel, knitr, lattice, maps, noncensus, shiny, testthat, tufte, xtable

SystemRequirements C++11

LinkingTo Rcpp (>= 1.0)

VignetteBuilder knitr

NeedsCompilation yes

BugReports https://github.com/cpanse/recmap/issues

Author Christian Panse [aut, cre] (<https://orcid.org/0000-0003-1975-3064>)

Repository CRAN

Date/Publication 2019-06-03 14:40:02 UTC

R topics documented:

`as.recmap.SpatialPolygonsDataFrame` *Convert a SpatialPolygonsDataFrame Object to recmap Object*

Description

The method generates a recmap class out of a `SpatialPolygonsDataFrame` object.

Usage

```
## S3 method for class 'SpatialPolygonsDataFrame'
as.recmap(X)
```

Arguments

`X`	a `SpatialPolygonsDataFrame` object.

Value

returns a `recmap` object.

Author(s)

Christian Panse

References

Roger S. Bivand, Edzer Pebesma, Virgilio Gomez-Rubio, 2013. Applied spatial data analysis with R, Second edition. Springer, NY. http://www.asdar-book.org/

See Also

- `sp`
- `as.SpatialPolygonsDataFrame`

Examples

```
  SpDf <- as.SpatialPolygonsDataFrame(recmap(checkerboard(8)))

  summary(SpDf)
  spplot(SpDf)

  summary(as.recmap(SpDf))



  # taken from https://github.com/sjewo/cartogram

## Not run:
     if (requireNamespace("maptools", quietly = TRUE)) {
library(maptools)
        data(wrld_simpl)

afr <- as.recmap(wrld_simpl[wrld_simpl$REGION==2, ])
is.recmap(afr)

afr$z <- afr$POP2005

is.recmap(afr)
afr <- afr[afr$z > 0, ]

# make map overlap to generate a connected pseudo dual
afr$dx <- 2.0 * afr$dx
afr$dy <- 2.0 * afr$dy


# overview
plot(recmap(afr))

# use the GA
set.seed(1)
plot(recmapGA(afr))

     }

## End(Not run)
```

`as.SpatialPolygonsDataFrame.recmap`
Convert a recmap Object to SpatialPolygonsDataFrame Object

Description

The method generates a SpatialPolygons object of a as input given `recmap` object. Both `data.frames` are merged by the index order.

Usage

```
## S3 method for class 'recmap'
as.SpatialPolygonsDataFrame(x, df = NULL, ...)
```

Arguments

x	a recmap object.
df	a data.frame object. default is NULL.
...	...

Value

returns a SpatialPolygonsDataFrame

Note

This S3method has replaced the recmap2sp function since package version 0.5.22.

Author(s)

Christian Panse

References

Roger S. Bivand, Edzer Pebesma, Virgilio Gomez-Rubio, 2013. Applied spatial data analysis with R, Second edition. Springer, NY. http://www.asdar-book.org/

See Also

- sp

Examples

```
SpDf <- as.SpatialPolygonsDataFrame(recmap(checkerboard(8)))
summary(SpDf)
spplot(SpDf)
```

---

checkerboard *Create a Checkboard*

Description

This function generates a recmap object.

Usage

```
checkerboard(n = 8, ratio = 4)
```

Arguments

n	defines the size of the map. default is 8 which will generate a map having 64 regions.
ratio	defines the ratio of the statistical value. As default, the black regions have a value which is four times higher.

Value

returns a checkerboard as recmap object.

Author(s)

Christian Panse

See Also

- recmap.

Examples

```
checkerboard8x8 <- checkerboard(8)

plot(checkerboard8x8,
  col=c('white','white','white','black')[checkerboard8x8$z])
```

---

jss2711 *jss2711 data*

Description

jss2711 contains the replication materials (input and output) for the jss.v086.c01 manuscript's Figures 4, 5, 6, 7, 11, 12, and 13.

Format

A set of nested list of data.frames.

Author(s)

Christian Panse, 2018

Source

- Figure 4 – `mbb_check` contains a `data.frame` with some `recmap` implemention benchmarks. Generated on
  - MacBook Pro (15-inch, 2017).
  - Processor: 2.9 GHz Intel Core i7
  - Memory: 16 GB 2133 MHz LPDDR3
- Figure 5 – `cmp_GA_GRASP` contains a `list` of results using a `GRASP` and `GA` metaheuristic. Generated on a MacBook Pro (Retina, 13-inch, Mid 2014).
- Figure 11 – `Switzerland`:
  - input map rectangles derived from: Swiss Federal Office of Topography `https://www.swisstopo.admin.ch` using Landscape Models / Boundaries GG25, downloaded 2016-05-01; Perfomed on a Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz/ Debian8
  - statistical data: Bundesamt fur Statistik (BFS) `https://www.bfs.admin.ch`, Website Statistik Schweiz, downloaded file `je-d-21.03.01.xls` on 2016-05-26.,
  - Perfomed on a Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz/ Debian8.
- Figure 12 – `SBB`:
  - Source: https://data.sbb.ch/explore 2016-05-12.
  - Perfomed on a Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz/ Debian 8.
- Figure 13 – `UK`:
  - input map rectangles derived from: `https://census.edina.ac.uk/ukborders`; Contains OS data Crown copyright [and database right] (2016);
  - Source of election data: NISRA
  - copyright - Contains National Statistics data Crown copyright and database right 2016 Contains NRS data Crown copyright and database right 2016
  - Perfomed on a Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz/ Debian8

References

Panse C (2018). "Rectangular Statistical Cartograms in R: The recmap Package." Journal of Statistical Software, Code Snippets, 86(1), pp. 1-27. doi: 10.18637/jss.v086.c01.

Examples

```
options(warn = -1)

## Figure 4
jss2711_figure4 <- function(nrep = 1, size = 2:10){
  recmap_debug_code <- '
  // [[Rcpp::plugins(cpp11)]]

  #include <Rcpp.h>
  #include <string>
  #include <recmap.h>

  using namespace Rcpp;
```

```
// [[Rcpp::depends(recmap)]]
// [[Rcpp::export]]
int recmap_debug(DataFrame df,
  bool map_region_intersect_multiset = true) {
  // access the columns
  NumericVector x = df["x"];
  NumericVector y = df["y"];
  NumericVector dx = df["dx"];
  NumericVector dy = df["dy"];


  NumericVector z = df["z"];
  CharacterVector name = df["name"];

  NumericVector cartogram_x(x.size());
  NumericVector cartogram_y(x.size());
  NumericVector cartogram_dx(x.size());
  NumericVector cartogram_dy(x.size());

  NumericVector dfs_num(x.size());
  NumericVector topology_error(x.size());
  NumericVector relpos_error(x.size());
  NumericVector relpos_nh_error(x.size());

  crecmap::RecMap X;
  X.set_map_region_intersect_multiset(map_region_intersect_multiset);

  for (int i = 0; i < x.size(); i++) {
    std::string sname = Rcpp::as<std::string>(name[i]);
    X.push_region(x[i], y[i], dx[i], dy[i], z[i],  sname);
  }

  X.run(true);

  return(X.get_intersect_count());
}
'

sourceCpp(code = recmap_debug_code, rebuild = TRUE, verbose = TRUE)

do.call('rbind', lapply(size, function(size){
  set.seed(1);
  CB <- checkerboard(size);

  do.call('rbind',lapply(rep(size, nrep), function(n){

    CB.smp <- CB[sample(nrow(CB), nrow(CB)), ]
    start_time <- Sys.time()
    ncall.multiset <- recmap_debug(CB.smp,
       map_region_intersect_multiset = TRUE)

    end_time <- Sys.time()
```

```
      diff_time.multiset <- as.numeric(difftime(end_time,
        start_time, units = "secs"))


      start_time <- Sys.time()
      ncall.list <- recmap_debug(CB.smp,
        map_region_intersect_multiset = FALSE)
      end_time <- Sys.time()
      diff_time.list <- as.numeric(difftime(end_time,
        start_time, units = "secs"))

      rv <- rbind(data.frame(number = ncall.multiset,
        algorithm="multiset", size = nrow(CB),
  time_in_secs = diff_time.multiset),
        data.frame(number = ncall.list,
  algorithm="list", size = nrow(CB),
    time_in_secs =  diff_time.list))

      rv$machine <- Sys.info()['machine']
      rv$sysname <- Sys.info()['sysname']
      rv
      }))
    }))
}

## Not run:
mbb_check <- jss2711_figure4()

## End(Not run)

data(jss2711)
boxplot(number  ~ sqrt(size),
  axes=FALSE,
  data = mbb_check,
  log='y',
  cex = 0.75,
  subset = algorithm == "list",
  col = "red", boxwex = 0.25);
abline(v = sqrt(50), col = 'lightgray', lwd = 3)

boxplot(number  ~ sqrt(size),
  data = mbb_check,log='y',
  subset = algorithm == "multiset",
  cex = 0.75,
  ylab = 'number of MBB intersection calls',
  xlab = 'number of map regions',
  boxwex = 0.25, add = TRUE, axes=FALSE);
axis(2)
axis(1, c(5, sqrt(50), 10, 15, 20), c("5x5", "US", "10x10", "15x15", "20x20"))
box()

legend("bottomright", c("C++ STL list", "C++ STL multiset"),
      col=c('red', 'black'), pch = 16, cex = 1.0)
```

```
## Figure 5

op <- par(mar=c(0, 0, 0, 0), mfrow=c(1, 3), bg = 'azure')

plot(cmp_GA_GRASP$GRASP$Map,
     border='black',
     col=c('white', 'white', 'white', 'black')[cmp_GA_GRASP$GRASP$Map$z])

plot(cmp_GA_GRASP$GRASP$Cartogram,
     border='black',
     col = c('white', 'white', 'white', 'black')[cmp_GA_GRASP$GRASP$Cartogram$z])

plot(cmp_GA_GRASP$GA$Cartogram,
     border='black',
     col = c('white', 'white', 'white', 'black')[cmp_GA_GRASP$GA$Cartogram$z])
par(op)

## Figure 6 - right

op <- par(mar = c(0, 0, 0, 0), mfrow=c(1, 1), bg = 'azure')
# found by the GA
smp <- cmp_GA_GRASP$GA$GA@solution[1,]

Cartogram.Checkerboard <- recmap(cmp_GA_GRASP$GA$Map[smp, ])
idx <- order(Cartogram.Checkerboard$dfs.num)

plot(Cartogram.Checkerboard,
     border='black',
     col=c('white', 'white', 'white', 'black')[Cartogram.Checkerboard$z])

# draw placement order
lines(Cartogram.Checkerboard$x[idx],
  Cartogram.Checkerboard$y[idx],
  col = rgb(1,0,0, alpha=0.3), lwd = 4, cex=0.5)

text(Cartogram.Checkerboard$x[idx],
  Cartogram.Checkerboard$y[idx],
  1:length(idx), pos=1, col=rgb(1,0,0, alpha=0.7))

points(Cartogram.Checkerboard$x[idx[1]],
  Cartogram.Checkerboard$y[idx[1]], lwd = 5, col = 'red')
text(Cartogram.Checkerboard$x[idx[1]],
  Cartogram.Checkerboard$y[idx[1]], "start", col = 'red', pos=3)
points(Cartogram.Checkerboard$x[idx[length(idx)]],
  Cartogram.Checkerboard$y[idx[length(idx)]],
       cex = 1.25, lwd = 2, col = 'red', pch = 5)
par(op)
op <- par(mar = c(4, 4, 1.5, 0.5), mfrow = c(1, 1), bg = 'white')
plot(best ~ elapsedtime, data = cmp_GA_GRASP$cmp,
     type = 'n',
```

```
     ylab = 'best fitness value',
     xlab = 'elapsed time [in seconds]')
abline(v=60, col='lightgrey',lwd=2)
lines(cmp_GA_GRASP$cmp[cmp_GA_GRASP$cmp$algorithm == "GRASP",
  c('elapsedtime', 'best')], type = 'b', col='red', pch=16)
lines(cmp_GA_GRASP$cmp[cmp_GA_GRASP$cmp$algorithm == "GA",
  c('elapsedtime', 'best')], type = 'b', pch=16)
legend("bottomright",
  c("Evolutionary based Genetic Algorithm (GA)",
    "Greedy Randomized Adaptive Search Procedures (GRASP)"),
    col = c('black', 'red'),
       pch=16, cex=1.0)

par(op)

## Figure 7
## Not run:

res <- lapply(c(1, 1, 2, 2, 3, 3), function(seed){
  set.seed(seed);
  res <- recmapGA(Map = checkerboard(4), pmutation = 0.25)
  res$seed <- seed
  res})

op <- par(mfcol=c(2,4), bg='azure', mar=c(5, 5, 0.5, 0.5))

x <- recmap(checkerboard(4))
p <- paste(' = (', paste(1:length(x$z), collapse=", "), ')', sep='')
plot(x,
      sub=substitute(paste(Pi['forward'], p), list(p=p)),
      col = c('white', 'white', 'white', 'black')[x$z])

x <- recmap(checkerboard(4)[rev(1:16),])
p <- paste(' = (', paste(rev(1:length(x$z)), collapse=", "), ')', sep='')
plot(x,
      sub=substitute(paste(Pi[reverse], p), list(p=p)),
      col = c('white', 'white', 'white', 'black')[x$z])


rv <- lapply(res, function(x){
  p <- paste(' = (', paste(x$GA@solution[1,], collapse=", "), ')', sep='')
  plot(x$Cartogram,
       col = c('white', 'white', 'white', 'black')[x$Cartogram$z],
       sub=substitute(paste(Pi[seed], perm), list(perm=p, seed=x$seed)))
  })

## End(Not run)

# sanity check - reproducibility

identical.recmap <- function(x, y, plot.diff = FALSE){
  target <- x
  current <- y
```

```
  stopifnot(is.recmap(target))
  stopifnot(is.recmap(current))
  rv <- identical(x$x, y$x) && identical(x$y, y$y)  &&
    identical(x$dx, y$dx) && identical(x$dy, y$dy)
  if (plot.diff){
   rvtemp <- lapply(c('x', 'y', 'dx', 'dy'), function(cn){
        plot(sort(abs(target[, cn] - current[, cn])),
          ylab = 'absolute error',
          main = cn)
        abline(h = 0, col = 'grey')
      })
  }

  rv
}

## Not run:
op <- par(mfcol = c(4, 4), mar = c(4, 4, 4, 1));
identical.recmap(res[[1]]$Cartogram, res[[2]]$Cartogram, TRUE)
identical.recmap(res[[3]]$Cartogram, res[[4]]$Cartogram, TRUE)
identical.recmap(res[[5]]$Cartogram, res[[6]]$Cartogram, TRUE)
identical.recmap(res[[1]]$Cartogram, res[[6]]$Cartogram, TRUE)

## End(Not run)

## Figure 11
## Not run: plot(recmap(Switzerland$map[Switzerland$solution,]))

op <- par(mfrow=c(1, 1), mar=c(0,0,0,0));

C <- Switzerland$Cartogram

plot(C)

idx <- rev(order(C$z))[1:50];

text(C$x[idx], C$y[idx], C$name[idx], col = 'red',
  cex = C$dx[idx] / strwidth(as.character(C$name[idx])))

## Figure 12

fitness.SBB <- function(idxOrder, Map, ...){
  Cartogram <- recmap(Map[idxOrder, ])
  if (sum(Cartogram$topology.error == 100) > 1){return (0)}
  1 / sum(Cartogram$z / (sqrt(sum(Cartogram$z^2))) * Cartogram$relpos.error)
}

## Not run:
SBB <- recmapGA(SBB$Map,
  parallel=TRUE,
  maxiter=1000,
  run=1000,
```

```
  seed = 1,
  keepBest = TRUE,
  fitness=fitness.SBB)

## End(Not run)

SBB.Map <- SBB$Map

# make input map overlapping
S <- SBB$Map
S <- S[!is.na(S$x),]
S$dx <- 0.1; S$dy <- 0.1; S$z <- S$DTV
S$name <- S$Bahnhof_Haltestelle

op <- par(mfrow = c(2, 1), mar = c(0, 0, 0, 0))
plot.recmap(S)
idx <- rev(order(S$z))[1:10]
text(S$x[idx], S$y[idx], S$name[idx], col='red', cex=0.7)
idx <- rev(order(S$z))[11:30]
text(S$x[idx], S$y[idx], S$name[idx], col = 'red', cex = 0.5)

Cartogram.recomp <- recmap(S)
plot(Cartogram.recomp)

idx <- rev(order(Cartogram.recomp$z))[1:40]
text(Cartogram.recomp$x[idx],Cartogram.recomp$y[idx],
Cartogram.recomp$name[idx],
col = 'red',
cex = 1.25 * Cartogram.recomp$dx[idx] / strwidth(Cartogram.recomp$name[idx]))

# sanity check - reproducibility cross plattform
op <- par(mfrow = c(2, 2), mar = c(5, 5, 5, 5))
identical.recmap(Cartogram.recomp, SBB$Cartogram, TRUE)


## Figure 13

## Not run:
DF <- data.frame(Pct_Leave = UK$Map$Pct_Leave, row.names = UK$Map$name)
spplot(as.SpatialPolygonsDataFrame(UK$Map, DF),
  main="Input England/Wales/Scottland")

UK.recmap <- recmap(UK$Map)
spplot(as.SpatialPolygonsDataFrame(UK.recmap , DF))

# sanity check - reproducibility cross plattform
op <- par(mfrow=c(2,2), mar=c(5,5,5,5))
identical.recmap(UK.recmap, UK$Cartogram, TRUE)

## End(Not run)
```

`plot.recmap` *Prints a recmap object.*

Description

plots input and output of the `recmap` function. The function requires column names (x, y, dx, dy).

Usage

```
## S3 method for class 'recmap'
plot(x, col='#00000011', col.text = 'grey', border = 'darkgreen', ...)
```

Arguments

`x`	`recmap` object - can be input or output of `recmap`.
`col`	a vector of colors.
`border`	This parameter is passed to the `rect` function. color for rectangle border(s). The default means par("fg"). Use border = NA to omit borders. If there are shading lines, border = TRUE means use the same colour for the border as for the shading lines. The default value is set to `'darkgreen'`.
`col.text`	a vector of colors.
`...`	whatsoever

Value

graphical output

Author(s)

Christian Panse

See Also

`recmap` or `vignette("recmap")`

Examples

```
Cartogram <- recmap(Map <- checkerboard(2))

plot(Map)
plot(Cartogram)
```

recmap *Compute a Rectangular Statistical Cartogram*

Description

The input consists of a map represented by overlapping rectangles. The algorithm requires as input for each map region:

- a tuple of (x, y) values corresponding to the (longitude, latitude) position,
- a tuple of (dx, dy) of expansion along (longitude, latitude),
- and a statistical value z.

The (x, y) coordinates represent the center of the minimal bounding boxes (MBB), The coordinates of the MBB are derived by adding or subtracting the (dx, dy) / 2 tuple accordingly. The tuple (dx, dy) defines also the ratio of the map region. The statistical values define the desired area of each map region.

The output is a rectangular cartogram where the map regions are:

- Non-overlapping,
- connected,
- ratio and area of each rectangle correspond to the desired areas,
- rectangles are placed parallel to the axes.

The construction heuristic places each rectangle in a way that important spatial constraints, in particular

- the topology of the pseudo dual graph,
- the relative position of MBB centers.

are tried to be preserved.

The ratios are preserved and the area of each region corresponds to the as input given statistical value z.

Usage

```
recmap(V, E = data.frame(u=integer(), v=integer()))
```

Arguments

V	defines the input map regions formatted as data.frame having the column names c('x', 'y', 'dx', 'dy', 'z', 'name') as described above. V could also be considered as the nodes of the pseudo dual.
E	defines the edges of the map region's pseudo dual graph. If E is not provided, this is the default; the pseudo dual graph is composed of overlapping rectangles. If used, E must be a data.frame containing two columns named c('u', 'v') of type integer referencing the row number of V.

Details

The basic idea of the current recmap *implementation*:

1. Compute the pseudo dual out of the overlapping map regions.
2. Determine the *core region* by `index <- int(n / 2)`.
3. Place region by region along DFS skeleton of pseudo dual starting with the *core region*.

Note: if a rectangle can not be placed, accept a non-*feasible solution* (avoid solutions having a topology error higher than 100) Solving this constellation can be intensive in the computation and due to the assumably low fitness value the candidate cartogram will be likely rejected by the metaheuristic.

Time Complexity: The time complexity is $O(n^2)$, where n is the number of regions. DFS is visiting each map region only once and therefore has time complexity $O(n)$. For each placement, a constant number of MBB intersection are called (max 360). MBB check is implemented using `std::set`, `insert`, `upper_bound`, `upper_bound` costs are $O(\log(n))$. However, worst case for a range query is $O(n)$, if and only if dx or dy cover the whole x or y range. Q.E.D.

Performance: In praxis, computing on a 2.4 GHz Intel Core i7 machine (using only one core), using the 50 state U.S. map example, recmap can compute approximately 100 cartograms in one second. The number of MBB calls were (Min., Median, Mean, Max) = (1448, 2534, 3174, 17740), using in each run a different index order using the (`sample`) method.

Geodetic datum: the `recmap` algorithm is not transforming the geodetic datum, e.g., WGS84 or Swissgrid.

Value

Returns a `recmap` S3 object of the transformed map with new coordinates (x, y, dx, dy) plus additional columns containing information for topology error, relative position error, and the DFS number. The error values are thought to be used for fitness function of the metaheuristic.

Author(s)

Christian Panse, 2016

References

- Roland Heilmann, Daniel Keim, Christian Panse, and Mike Sips (2004). "RecMap: Rectangular Map Approximations." InfoVis 2004, IEEE Symposium on Information Visualization, Austin, Texas, 33-40. doi: 10.1109/INFVIS.2004.57.
- Panse C (2018). "Rectangular Statistical Cartograms in R: The recmap Package." Journal of Statistical Software, Code Snippets, 86(1), pp. 1-27. doi: 10.18637/jss.v086.c01.

See Also

- S3 class methods: `plot.recmap` and `summary.recmap`.
- the metaheuristic glue methods: `recmapGA` and `recmapGRASP`.
- The package vignette `https://CRAN.R-project.org/package=recmap/vignettes/recmap.html` or by calling `vignette("recmap")` on the R command prompt.

Examples

```
map <- checkerboard(2)
cartogram <- recmap(map)

map
cartogram

op <- par(mfrow = c(1, 2))
plot(map)
plot(cartogram)

## US example
usa <- data.frame(x = state.center$x,
  y = state.center$y,
  # make the rectangles overlapping by correcting
  # lines of longitude distance.
  dx = sqrt(state.area) / 2
    / (0.8 * 60 * cos(state.center$y * pi / 180)),
  dy = sqrt(state.area) / 2 / (0.8 * 60),
  z = sqrt(state.area),
  name = state.name)

usa$z <- state.x77[, 'Population']
US.Map <- usa[match(usa$name,
  c('Hawaii', 'Alaska'), nomatch = 0)  == 0, ]

plot.recmap(US.Map)
plot(recmap(US.Map))
par(op)

# define a fitness function
recmap.fitness <- function(idxOrder, Map, ...){
  Cartogram <- recmap(Map[idxOrder, ])
  # a map region could not be placed;
  # accept only feasible solutions!
  if (sum(Cartogram$topology.error == 100) > 0){return (0)}
  1 / sum(Cartogram$z / (sqrt(sum(Cartogram$z^2)))
    * Cartogram$relpos.error)
}


## Not run:

## use Genetic Algorithms (GA >=3.0.0) as metaheuristic

M <- US.Map

recmapGA <- ga(type = "permutation",
  fitness = recmap.fitness,
  Map = M,
  monitor = gaMonitor,
  min = 1, max = nrow(M) ,
```

```
  popSize = 4 * nrow(M),
  maxiter = 10,
  run = 100,
  parallel=TRUE,
  pmutation = 0.25)

plot(recmap(M[recmapGA@solution[1,],]))

plot(recmapGA)


## End(Not run)


## Not run:
# install.packages('rbenchmark')

library(rbenchmark)
benchmark(recmap(US.Map[sample(50,50),]), replications=100)
##                              test replications elapsed relative user.self
##1 recmap(US.Map[sample(50, 50), ])          100   1.255        1     1.124
##  sys.self user.child sys.child
##1    0.038          0         0

## End(Not run)

## Have Fun!
```

---

recmapGA *Genetic Algorithm Wrapper Function for recmap*

Description

higher-level function for recmap using a Genetic Algorithm as metaheuristic.

Usage

```
recmapGA(Map,
  fitness = .recmap.fitness,
  pmutation = 0.25,
  popSize = 10 * nrow(Map),
  maxiter = 10,
  run = maxiter,
  monitor = if (interactive()) { gaMonitor } else FALSE,
  parallel = FALSE,
  ...)
```

Arguments

	The following arguments are passed through ga.
Map	defines the input map regions formatted as data.frame having the column names c('x', 'y', 'dx', 'dy', 'z', 'name') as described above.
fitness	a fitness function function(idxOrder, Map, ...) returning a number which as to be maximized.
pmutation	see docu of ga.
popSize	see docu of ga.
maxiter	see docu of ga.
run	see docu of ga.
monitor	see docu of ga.
parallel	see docu of ga.
...	passed through the ga method.

Value

returns a list of the input Map, the solution of the ga function, and a recmap object containing the cartogram.

Author(s)

Christian Panse 2016-2019

References

Luca Scrucca (2013). GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37. http://dx.doi.org/10.18637/jss.v053.i04.

See Also

- recmap - Compute a Rectangular Statistical Cartogram
- ga - Genetic Algorithms

Examples

```
## The default fitnes function is currently defined as
function(idxOrder, Map, ...){

  Cartogram <- recmap(Map[idxOrder, ])
  # a map region could not be placed;
  # accept only feasible solutions!

  if (sum(Cartogram$topology.error == 100) > 0){return (0)}

  1 / sum(Cartogram$relpos.error)
}
```

```
## use Genetic Algorithms (GA >=3.0.0) as metaheuristic
set.seed(1)
res <- recmapGA(Map = checkerboard(4), pmutation = 0.25)

op <- par(mfrow = c(1, 3))
plot(res$Map, main = "Input Map")
plot(res$GA, main="Genetic Algorithm")
plot(res$Cartogram, main = "Output Cartogram")


## US example
getUS_map <- function(){
  usa <- data.frame(x = state.center$x,
  y = state.center$y,
  # make the rectangles overlapping by correcting
  # lines of longitude distance.
  dx = sqrt(state.area) / 2
    / (0.8 * 60 * cos(state.center$y * pi / 180)),
  dy = sqrt(state.area) / 2 / (0.8 * 60),
  z = sqrt(state.area),
  name = state.name)

  usa$z <- state.x77[, 'Population']
  US.Map <- usa[match(usa$name,
    c('Hawaii', 'Alaska'), nomatch = 0)  == 0, ]

  class(US.Map) <- c('recmap', 'data.frame')
  US.Map
}

## Not run:
# takes 34.268 seconds on CRAN
res <- recmapGA(getUS_map(), maxiter = 5)
op <- par(ask = TRUE)
plot(res)
par(op)
summary(res)

## End(Not run)
```

---

recmapGRASP *Greedy Randomized Adaptive Search Procedure Wrapper Function for recmap*

Description

Implements a metaheuristic for recmap based on GRASP.

Usage

```
recmapGRASP(Map, fitness = .recmap.fitness, n.samples = nrow(Map) * 2,
  fitness.cutoff = 1.7, iteration.max = 10)
```

Arguments

Map	defines the input map regions formatted as data.frame having the column names c('x', 'y', 'dx', 'dy', 'z', 'name') as described above.
fitness	a fitness function function(idxOrder, Map, ...) returning a number which as to be maximized.
n.samples	number of samples.
fitness.cutoff	cut-off value.
iteration.max	maximal number of iteration.

Value

returns a list of the input Map, the best solution of GRASP, and a recmap object containing the cartogram.

Author(s)

Christian Panse

References

Feo TA, Resende MGC (1995). "Greedy Randomized Adaptive Search Procedures." Journal of Global Optimization, 6(2), 109-133. ISSN 1573-2916. doi: 10.1007/BF01096763.

See Also

recmapGA and recmap

Examples

```
## US example
getUS_map <- function(){
  usa <- data.frame(x = state.center$x,
  y = state.center$y,
  # make the rectangles overlapping by correcting
  # lines of longitude distance.
  dx = sqrt(state.area) / 2
    / (0.8 * 60 * cos(state.center$y * pi / 180)),
  dy = sqrt(state.area) / 2 / (0.8 * 60),
  z = sqrt(state.area),
  name = state.name)

  usa$z <- state.x77[, 'Population']
  US.Map <- usa[match(usa$name,
```

```
    c('Hawaii', 'Alaska'), nomatch = 0)  == 0, ]

  class(US.Map) <- c('recmap', 'data.frame')
  US.Map
}

## Not run:
res <- recmapGRASP(getUS_map())
plot(res$Map, main = "Input Map")
plot(res$Cartogram, main = "Output Cartogram")

## End(Not run)
```

`summary.recmap` *Summary for recmap*

Description

Summary method for S3 class `recmap`. The area error is computed as described in the CartoDraw paper.

Usage

```
## S3 method for class 'recmap'
summary(object, ...)
```

Arguments

`object`	S3 class `recmap`.
`...`	whatsoever.

Value

returns a `data.frame` containing summary information, e.g., objective functions or number of map regions.

Author(s)

Christian Panse 2016

References

D. A. Keim, S. C. North, and C. Panse, "CartoDraw: A Fast Algorithm for Generating Contiguous Cartograms," IEEE Trans. Vis. Comput. Graph., vol. 10, no. 1, pp. 95-110, 2004. doi: 10.1109/TVCG.2004.1260761.

Examples

```
summary(checkerboard(4))
summary(recmap(checkerboard(4)))
```

Index