

Package ‘BARD’

February 9, 2012

Type Package

Title Better Automated ReDistricting

Version 1.24

Date 2011-04-29

Author Micah Altman <Micah_Altman@harvard.edu>

Maintainer Micah Altman <Micah_Altman@harvard.edu>

Description This is a package for automated redistricting and heuristic exploration of redistricter revealed preference

Depends R (>= 2.10), digest

Imports maptools,spdep

Suggests rgenoud,multicore,rJava,iplots

Enhances gpclib,R2HTML,rgdal,snow,ineq

License AGPL-3 + file LICENSE

URL <http://bard.sf.net>

Repository CRAN

Date/Publication 2011-05-03 15:27:08

R topics documented:

BARD-package	2
basem<-.default	5
calcLWCompactScore	6
choroplexPlan	10
combineDynamicScores	11
createRandomPlan	13
editPlanInteractive	17

fillHolesPlan	19
hplot	20
iacounty.sdf	22
importBardShape	22
importBlockEquiv	24
miniball	25
nb2graph	26
PMPreport	27
profilePlans	29
readBardCheckpoint	32
readBardImage	34
readBardMap	35
refineGreedyPlan	36
reportPlans	39
scorePlans	41
spatialDataFrame2bardBasemap	42
startBardCluster	43
suffolk.map	45

Index	46
--------------	-----------

BARD-package	<i>A package for better automated redistricting.</i>
--------------	--

Description

BARD will automatically generate redistricting plans using multi-criteria optimization algorithms. BARD can analyze and compare plans for differences in assignment and criteria. BARD supports heuristic exploration of plans, in order to show trade-offs among redistricting criteria.

Details

Package:	BARD
Type:	Package
Version:	1.10
Date:	2010-4-24
License:	AGPL 3.0

Bard supports several areas of functionality: 1. Plan input output. Reading and writing plans in various formats. ([importBlockEquiv](#), [linkreadBardImage](#), [readBardCheckpoint](#), [exportBardShape](#),[readBardMap](#),[writeBardMap](#))

2. Initial plan generation.

Quick heuristics for generating random plans, or plans based on a fixed set of criteria. ([createRandomPlan](#), [createRandomPopPlan](#), [createContiguousPlan](#), [createKmeansPlan](#), [createWeightedKmeansPlan](#), [createGreedyContiguousPlan](#))

Heuristics for fixing contiguity and unassigned blocks. ([fixContiguityPlan](#), [fixUnassignedPlan](#))

As a quick method for exploration, single districts can be pseudo-sampled. ([createContiguousDistrict](#), [quickSampleDistricts](#))

3. Interactive plan editing. Adjust plans interactively, using a mouse. ([editPlanInteractive](#), [createPlanInteractive](#))

4. Plan scoring. Scoring functions for use in plan refinement, profiling, and comparison. ([calcContiguityScore](#), [calcReockScore](#), [calcPACompactScore](#), [calcLWCompactScore](#), [calcGroupScore](#), [calcPopScore](#), [calcSpatialHolesScore](#), [calcUnassignedScore](#), [calcRangeScore](#), [calcMomentScore](#), [calcBBCompactScore](#), [combineDynamicScores](#), [calcSplitScore](#), [calcIneqScore](#))

5. Plan refinement. Multi-criteria optimization heuristics for refining plans to meet specified goals. ([refineGreedyPlan](#), [refineAnnealPlan](#), [refineGenoudPlan](#), [refineNelderPlan](#), [refineGRASPPlan](#), [startBardCluster](#), [refineTabuPlan](#))

6. Plan sampling, profiling and exploration. Generate profiles of plans to explore tradeoffs among redistricting criteria. This can be used in conjunction with *snow* to distribute plan generation across a computing cluster, or with *multicore* to run in parallel across multiple cores on a single system.

([samplePlans](#), [quickSampleDistricts](#), [profilePlans](#), [startBardCluster](#))

8. Plan comparison and analysis. Plot, score, and compare plans. Create HTML reports. ([reportPlans](#), [scorePlans](#), [diff.bardPlan](#), [choroplexPlan](#), [PMPreport](#))

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

Maintainer: Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Altman \& McDonald, 2004. A computation intensive method for Evaluating Intent in Redistricting. Presented at the Midwest Political Science Association Meeting, Micah Altman, 1997. “Is Automation the Answer? The Computational Complexity of Automated Redistricting”, Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

C. Cirincione , T.A. Darling, and T.G. O’Rourke. 2000. “Assessing South Carolina’s 1990’s Congressional Districting.” *Political Geography* 19: 189-211.

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. “From Crayons to Computers: The Evolution of Computer Use in Redistricting” *Social Science Computer Review* 23(3): 334-46.

See Also

[spdep](#) , [readShapePoly](#) , [genoud](#)

Examples

```

suffolk.map <- importBardShape(
  file.path(system.file("shapefiles", package="BARD"),"suffolk_tracts")
)

numberdists <- 5
kplan <- createKmeansPlan(suffolk.map,numberdists)
rplan <- createRandomPlan(suffolk.map,numberdists)
rplan2 <- createRandomPopPlan(suffolk.map,numberdists)
plot(kplan)

reportPlans(plans=list("kmeans"=kplan,"random 1"=rplan,"random pop"=rplan2), doplot=TRUE)

## Not run:
if (require("iplots",quietly=TRUE)) {
rplan<-editPlanInteractive(rplan,calcPopScore,predvar="POP")
}

## End(Not run)

# district sampling - quick

randomDists<-quickSampleDistricts(10,suffolk.map,numberdists)
distscores<- scorePlans(randomDists,scoreFUNs=list("LWCompact"=calcLWCompactScore,"PACompact"=calcPACompactScore)
plot(distscores[2:3])

myScore<-function(plan,...) {
  return(calcContiguityScore(plan,...))
}

#just for quick demonstration -- nelder method not effective

improvedRplan<-refineNelderPlan(plan=rplan2, score.fun=myScore, displaycount=100, historysize=0, dynamicscoring=FALSE)

## Not run:
# This works better, but will take a while
improvedRplan<-refineAnnealPlan(plan=rplan2, score.fun=myScore, historysize=0, dynamicscoring=FALSE, tracelevel=1)

## End(Not run)

samples<-samplePlans(kplan, score.fun=myScore, ngenplans=10, gen.fun = "createRandomPlan", refine.fun="refineNelderPlan")

profplans<-profilePlans( list(kplan,rplan), score.fun=calcContiguityScore, addscore.fun=calcPopScore, numevals=100)

summary(samples)
plot(summary(samples))
reportPlans(samples)
plot(summary(profplans))

```

basem<-.default	<i>manipulate plan basemaps</i>
-----------------	---------------------------------

Description

Set/get the basemap of a plan

Usage

```
basem(object, ...)  
"basem<-"(object, ..., value)  
"basem<-.default"(object, ..., value)  
basem.default(object, ...)
```

Arguments

object	plan
...	not used, for other generics
value	basemap

Value

returns basemap

Note

These methods are only for developers writing new score functions only.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

Examples

```
# read in a shapefile with demographic data  
data(suffolk.map)  
kplan<-createRandomPlan(suffolk.map,5)  
copy.of.suffolk.map<-basem(kplan)  
copy.of.suffolk.map==suffolk.map  
dim(copy.of.suffolk.map)  
  
# data frame w/out geography  
  
suffolk.data<-as.data.frame(suffolk.map)
```

calcLWCompactScore *Scoring functions for redistricting.*

Description

These functions evaluate redistricting plans.

Usage

```
calcRangeScore(plan, predvar="BLACK", predvar2="WHITE", targrange=c(.65,.70),
  sumdenom = TRUE,lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcGroupScore(plan,groups=list(),penalties=1,lastscore=NULL, changelist=NULL, standardize=TRUE)
```

```
calcPopScore(plan, predvar="POP",lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcPACompactScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcLWCompactScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcReockScore(plan, usebb=FALSE, lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcContiguityScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcBBCompactScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
```

```
calcUnassignedScore(plan, lastscore = NULL, changelist = NULL, standardize = TRUE)
```

```
calcHolesScore(plan, lastscore = NULL, changelist = NULL, standardize = TRUE)
```

```
calcSpatialHolesScore(plan, lastscore = NULL, changelist = NULL, standardize = TRUE)
```

```
calcMomentScore(plan,standardize=TRUE,centers=NULL,weightVar=NULL,penaltyExp=2, normalize=TRUE, las
```

```
calcSplitScore(plan, splitvar, lastscore = NULL, changelist = NULL, standardize = TRUE)
```

```
calcIneqScore(plan, eqvar, weightVar = NULL, parameter = NULL, type =c("Gini", "RS", "Atkinson", "Theil
```

Arguments

plan	plan to be scored
lastscore	optional, previous value returned by function, for incremental evaluation
changelist	Optional, a two column matrix of (column1) block ID's that were changed since lastscore was computed, (column2) previous plan assignments for those blocks
standardize	logical, should scores be standardized
penalties	penalties for splitting groups

predvar	name of variable in the basemap associated with plan
predvar2	name of second variable in the basemap associated with plan
targrange	acceptable target range for ratio of predictive variables
groups	a list of vectors, each vector should comprise the ids of the blocks in that group, groups may overlap
penaltyExp	a single number or vector of penalties for splitting each of the enumerated groups
centers	optional centers to use for calculating moment of inertia score, if absent, score is calculated using (weighted) district centroid (centroid of block centroids)
weightVar	name of variable in basemap to use for weighting
sumdenom	For calcRangeScore, whether to use the sum of both variables as the denominator, or only use the second variable
usebb	For calcReockScore use bounding boxes, instead of the entire polygon. This is a fast approximation, which is accurate where the polygons are much smaller than the district.
normalize	Flag – normalize moment of inertia score
splitvar	Variable containing geographic id, such as county id
eqvar	variable to be used for homogeneity score
type	type of inequality measure – see ineq
parameter	parameter for inequality measuer – see ineq

Details

calcContiguityScore- returns a score based on the number of separate contiguous regions in the district. The ideal district comprises a single contiguous region.

calcLWCompactScore – Returns a compactness score based on the ratio of the sides of the bounding rectangle for the district.

calcReockScore - Returns a compactness score based on the ratio of area to area of a circle. The usebb option uses polygon bounding boxes, which is much faster, and accurate if the polygons are small relative to the district.

calcBBCompactScore - Returns a compactness score based on the ratio of area to area of bounding box. Similar to Reock but much faster

calcPopScore - Returns a score based on the deviation from population equality of the districts.

calcRangeScore - Calculates districts compliance to a target range for a predictive variable. Will penalize districts increasingly as $\frac{\text{sum(predvar1)}}{\text{sum(predvar1)+sum(predvar2)}}$ (if sumdenom==TRUE) falls outside the given target range. Use this for majority-minority districts, partisan districts, competitive districts

calcGroupScore - Calculates plans compliance with keeping designated groups. Use for designated "nesting" districts, or known communities of interest. Returns proportion of group split by a district (if a group is split across multiple districts, all districts are penalized).

calcUnassignedScore - Penalizes plans for having unassigned blocks calcHolesScore - alias for calcUnassignedScore

calcSpatialHolesScore - penalizes plans for number of spatial (donut) holes (i.e. topological genus) in contiguous portions of the plan. (Completely separate portions of plans are not considered to have holes – but are counted by calcContiguityScore.

calcPACompactScore - calculates plan compactness as a perimeter/ratio area (where a Circle has a perfect score of 0)

calcMomentScore - calculates moment of inertia compactness score of the district, the sum or squared distance of the block centroids (weighted by area of the block) to the district centroids. Options allow changing the penalty exponent (e.g. penaltyExp=1 will sum distances), assign a weightVar which will be used instead of area (e.g. population, for population moment of inertia), or specify fixed district centers to use as a substitute for district centroid (e.g. to use for warehouse allocation problems). The normalization score gives the unitless, normalized version which is insensitive to scale (a default in version 1.17)

Value

All current plan score functions, except calcUnassignedScore return a vector of score value, representing the score for each district, with the plan score being the sum of this vector.

CalcUnassignedScore returns a single score for the plan. User-written score functions MAY also return a single number as a plan score instead, all bard utilities will handle this case correctly.

If the "standardize" option is true. Each of these values should be standardized to [0,1], with 0 representing the "best" score and 1 the worst score. Otherwise, score values MAY return values in other scalar ranges, and even invert the ranking. It is recommended that standardized scores be used except for testing.

Note that in order to support dynamic recalculation, the score vector returned may have additional attached attributes.

warning

calcSpatialHoles score will produce correct results only where the underlying districting units do not have spatial holes that fully contain other units. This is usually true of census blocks but not census tracts.

Note

All of the bard score functions implement some sort of dynamic update for efficiency when scoring large plans. The refinement methods make use of these dynamic score functions. The score can be dynamically recalculated based on the last score returned, accompanied by a list of changes. Dynamic recalculation is optional. Note that in order to support dynamic recalculation, the score vector returned may have additional attached attributes.

Incremental updating is not required of user-supplied score functions. Bard functions will check for the existence of a lastscore argument to determine whether this is available.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

- Micah Altman, 1997. “Is Automation the Answer? The Computational Complexity of Automated Redistricting”, Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml
- Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.
- C. Cirincione , T.A. Darling, and T.G. O’Rourke. 2000. “Assessing South Carolina’s 1990’s Congressional Districting.” *Political Geography* 19: 189-211.
- Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html
- Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. “From Crayons to Computers: The Evolution of Computer Use in Redistricting” *Social Science Computer Review* 23(3): 334-46.
- Altman, Micah, 1998. Districting Principles and Democratic Representation, Doctoral Thesis, California Institute of Technology.
- Niemi, R. G.; Grofman, B.; Carlucci, C.; and Hofeller T., 1990. Measuring Compactness and the Role of a Compactness Standard in a Test for Partisan and Racial Gerrymandering *Journal of Politics* 22:4 1155-1181.

See Also

- Plan refinement algorithms [refineGreedyPlan](#), [refineAnnealPlan](#), [refineGenoudPlan](#), [refineTabuPlan](#), [refineNelderPlan](#), .
- Combining dynamic scores [combineDynamicScores](#).
- Inequality package [ineq](#)

Examples

```
# read in a shapefile with demographic data
data(suffolk.map)
ndists<-5

# create some initial plans
kplan <- createKmeansPlan(suffolk.map,ndists)
rplan <- createRandomPlan(suffolk.map,ndists)
wkplan<-createWeightedKmeansPlan(suffolk.map,ndists,weightVar="POP")

calcPopScore(kplan)
calcPopScore(rplan)
calcLWCompactScore(rplan)
calcLWCompactScore(kplan)
calcPACompactScore(rplan)
calcPACompactScore(kplan)
calcUnassignedScore(kplan)
calcUnassignedScore(rplan)
calcSpatialHolesScore(kplan)
```

```

calcContiguityScore(rplan)
calcContiguityScore(kplan)
calcRangeScore(kplan)
calcRangeScore(rplan)
calcRangeScore(kplan, targrange=c(.01,.99))
calcRangeScore(rplan, targrange=c(.01,.99))
calcGroupScore(kplan,groups=list(c(1:10),c(100:120)),penalties=c(1,2))
calcGroupScore(rplan,groups=list(c(1:10),c(100:120)),penalties=c(1,2))
kplan2<-kplan1<-kplan
c1<-cbind(c(318,320),c(kplan1[318],kplan1[320]))

if ( ! all(calcPopScore(kplan2,lastscore=calcPopScore(kplan1),changelist=c1) == calcPopScore(kplan2)) ) {
  warning("dynamic score does not match!")
}

calcPopScore(kplan,predvar="POP")
calcPopScore(wkplan,predvar="POP")
calcMomentScore(kplan)
calcMomentScore(wkplan)
calcMomentScore(kplan,weightVar="POP")
calcMomentScore(wkplan,weightVar="POP")

```

choroplethPlan *Create a choropleth (shaded thematic map) plot for a redistricting plan*

Description

Plots a shaded thematic map on a plan, based on a set of score

Usage

```

choroplethPlan(plan, scores, numlevels=5,
method=c("quant", "equal", "absolute"),
main="choropleth map", absmin=0, absmax=1, ramplow="blue", ramphigh="red", ...)

```

Arguments

plan	plan to plot
scores	vector of scores
method	binning method
main	main title of plot
numlevels	number of bins
absmin	if method is absolute minimum for absolute range
absmax	if method is absolute maximum for absolute range

ramplow	color ramp endpoint for negative values
ramphigh	color ramp endpoint for positive values
...	arguments to pass on to map plotting

Details

The scores should represent values for each district in the plan. Each score is assigned to one of `numlevels` bins. Districts are plotted according to the bin they are assigned to, using a color ramp (or a double color ramp for ranges that extend from negative to positive).

Three methods of bin construction are supported: - `absolute`: creates equally spaced bins from `absmin` to `absmax` - `equal`: creates equally spaced bins from `min(scores)` to `max(scores)` - `quant`: bins are created using quantiles

Value

Nothing. The function is used for printing and plotting effect.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Scoring functions: [calcContiguityScore](#)

Examples

```
data(suffolk.map)

# choose number of districts
ndists <- 5

# create some initial plans
kplan <- createKmeansPlan(suffolk.map, ndists)
choroplethPlan(kplan, calcPopScore(kplan), numlevels=3)
```

combineDynamicScores *Convenience function for combining score functions.*

Description

This convenience function combines multiple dynamic score functions, while doing the housekeeping to track dynamic updates across each function.

Usage

```
combineDynamicScores(plan, lastscore=NULL, changelist=NULL, scorefuns=list(),
  distcombfun=sum, scorecombfun=sum)
```

Arguments

plan	plan to be scored
lastscore	optional, previous value returned by function, for incremental evaluation
changelist	Optional, a two column matrix of (column1) block ID's that were changed since lastscore was computed, (column2) previous plan assignments for those blocks
scorefuns	list of score functions
distcombfun	function used to combine scores across districts
scorecombfun	function for combining scores into a single score

Details

All of the bard score functions implement some sort of dynamic update for efficiency when scoring large plans. The refinement methods make use of these dynamic score functions. The score can be dynamically recalculated based on the last score returned, accompanied by a list of changes. Dynamic recalculation is optional. Note that in order to support dynamic recalculation, the score vector returned may have additional attached attributes.

This function combined multiple dynamic score functions correctly – by multiplexing and demultiplexing the attributes, so that each score function gets the appropriate tracking information.

Value

Returns a single score for the plan, plus attributes for dynamic recalculation.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> <http://maltman.hmdc.harvard.edu/>

References

Micah Altman, 1997. “Is Automation the Answer? The Computational Complexity of Automated Redistricting”, Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

C. Cirincione , T.A. Darling, and T.G. O'Rourke. 2000. “Assessing South Carolina’s 1990’s Congressional Districting.” *Political Geography* 19: 189-211.

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. “From Crayons to Computers: The Evolution of Computer Use in Redistricting” *Social Science Computer Review* 23(3): 334-46.

See Also

Plan refinement algorithms [refineGreedyPlan](#), [refineAnnealPlan](#), [refineGenoudPlan](#), [refineNelderPlan](#), [refineTabuPlan](#)

Examples

```
# read in a shapefile with demographic data
data(suffolk.map)
ndists<-5

# create some initial plans
kplan <- createKmeansPlan(suffolk.map,ndists)

calcPopScore(kplan)
calcLWCompactScore(kplan)
calcRangeScore(kplan, targrange=c(.01,.99))

combinedynamicScores(kplan,scorefuns=list(calcPopScore,calcLWCompactScore))

myScore<-function(plan,...){
  combinedynamicScores(plan,scorefuns=list(
    calcPopScore,
    calcLWCompactScore,
    calcContiguityScore,
    function(x,...)calcRangeScore(x,targrange=c(.01,.99),...)
  ))
}

myScore(kplan)

kplan2<-kplan1<-kplan
c1<-cbind(c(318,320),c(kplan1[318],kplan1[320]))
if ( ! all(calcPopScore(kplan2,lastscore=calcPopScore(kplan1),changelist=c1) == calcPopScore(
kplan2))) {
  warning("dynamic score does not match!")
}

kplan2[318]<-(kplan1[318]+1)
kplan2[320]<-(kplan1[320]+1)

oldscore<-myScore(kplan1)
if ( ! all (myScore(kplan2) == myScore(kplan2,lastscore=oldscore,changelist=c1))){
  warning("dynamic score does not match!")
}
```

Description

These methods create plans starting with random seeds. These plans should be refined with one of the refinement functions.

Usage

```
createRandomPlan(basemap, ndists)
createRandomPopPlan(basemap, ndists, predvar="POP")
createAssignedPlan(basemap, predvar="BARDPlanID", nseats=NULL, magnitudes=NULL)
createKmeansPlan(basemap, ndists)
createWeightedKmeansPlan( basemap, ndists, centers = c(), weightVar = NULL,
                           trimfactor = 2.5, smallBlock = c("cap", "closest"))
createContiguousPlan(basemap, ndists, predvar = "POP", threshold = 0.05,
                     ssize = 20, usebb = TRUE, maxtries = (10/threshold),
                     neighborstarts = TRUE, fillholes = TRUE, districtonly=FALSE, traceLevel=0)
createContiguousDistrict(basemap, ndists, predvar = "POP", threshold = 0.05,
                         ssize = 20, usebb = TRUE, maxtries = (10/threshold),
                         neighborstarts = TRUE, fillholes = TRUE)
createGreedyContiguousPlan( basemap, ndists, predvar="POP", scoreFUN=calcPopScore, display=FALSE, ...)
```

Arguments

basemap	Bard basemap from which to create plan
ndists	number of districts to create
predvar	name of variable of interest (for createAssignedPlan can be a vector or block assignments)
threshold	tolerance threshold for districts being "equal" in population
ssize	check up to ssize neighbors at random, choosing the one with the fewest foreign neighbors
usebb	use bounding box compactness
maxtries	maximum number of tries to create plan
neighborstarts	whether to start new district at neighbors of existing districts
fillholes	fill any holes remaining
centers	suggested centers for weighted kmeans
weightVar	name of weighting variable in basemap, for weighting
trimfactor	rescales and trims weights, see below
smallBlock	how weighted kmeans handles blocks that are weighted to 0: caps minimum weights at 1; closest post-assigns zero-weighted blocks to the nearest district
districtonly	whether to generate a single district of the plan only
scoreFUN	Bard score function
display	whether to display plans while creating – for demos and teaching
traceLevel	for debugging, tracelevel > 2 will turn on dynamic plotting
nseats	total seats (elected officials) represented by plan – for partial assignments

magnitudes a vector of length equal to the number of different districts in the assigned plan, with magnitudes of all districts – for multimember plans only

... arguments for score function

Details

createKmeansPlan create plans through applying kmeans to the center of each polygon. It tends toward contiguity, but is not guaranteed to produce contiguous plans

createRandomPlan creates plans by random block assignment, as per Grofman [1982], these are highly non-contiguous

createRandomPopPlan creates plans by random block assignment, but caps each district at a population threshold. It will not add a block to a district in formation that will cause the plan to go more than $(1+\text{threshold})/(\text{totalpopulation}/\text{plan})$, unless that block would cause all districts to exceed their thresholds. With blocks of single individuals, and a threshold of 0 this is equivalent to the random assignment discussed in Bush v. Vera [2004]

createGreedyContiguousPlan uses createRandomPopPlan followed by fixContiguityPlan

createContiguousPlan duplicates the algorithm in Cirincione, et. al (see the references). As per pg 196:

“The first algorithm, the contiguity algorithm, begins by randomly selecting a block group to serve as the *base* of the first district. It then constructs a *perimeter list* containing the unassigned block groups contiguous to the base block group. The program then randomly selects a block group from the perimeter list to add to the emerging district and adjusts the perimeter list. This process continues until the population of the emerging district achieves the desired district population level. (A newly created district is thrown out if its population deviates by more than 1 from the ideal district average population, which in this case is 581,117.) The next district begins with the random selection of a census block group from among those that touch one of the completed districts.”

(Note that ssize was not included as an option, originally)

In practice this method sometimes fails because creation of earlier districts prevents the creation of contiguous later districts. In this case, the method restarts, up to the maximum number of restarts. If fillholes is true, then even a final failed attempt will be patched with fixUnassignedPlan and fixContiguousPlan, however, beware that this may yield some districts as empty, or districts out of the target criteria range. In this case a warning is issued.

The districtonly option will cause a single district only to be created, which is a quick-and-dirty way of pseudo-sampling districts.

The resulting plan object is flagged as a ‘single-district’ plan and score functions will return the score for that district, not the plan as a whole.

createContiguousDistrict is a wrapper for createContiguousPlan with the districtonly option set.

createWeightedKmeansPlan weights kmeans by a particular variable, most often used with population. This is implemented through block replication, so to limit memory use, the weights are rescaled so that minimum and maximum weights span approximately trimfactor orders of magnitude. Zero-weight blocks can either be capped at 1 and incorporated in the kmeans calculation, or left out and assigned to the closest district, based on the value of smallblock.

Value

Returns a bard plan. Note that this plan is not guaranteed to be contiguous, or equipopulous.

Note

Turning on dynamic plotting is great for teaching and demos, but slows down plan creation by several orders of magnitude.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Micah Altman, 1997. "Is Automation the Answer? The Computational Complexity of Automated Redistricting", Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

Bush v. Vera, 517 U.S. 952 (1996).

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

C. Cirincione , T.A. Darling, and T.G. O'Rourke. 2000. "Assessing South Carolina's 1990's Congressional Districting." *Political Geography* 19: 189-211.

Grofman, B. 1982, "For single Member Districts Random is Not Equal", In *Representation and Redistricting Issues*, ed. B. Grofman, A. Lijphart, R. McKay, H. Scarrow. Lexington, MA: Lexington Books.

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. "From Crayons to Computers: The Evolution of Computer Use in Redistricting" *Social Science Computer Review* 23(3): 334-46.

See Also

Interactive editing [editPlanInteractive](#) Plan refinement algorithms [refineGreedyPlan](#), [refineAnnealPlan](#), [refineGenoudPlan](#), [refineNelderPlan](#). Plan fixups [fixUnassignedPlan](#) , [fixContiguityPlan](#)

Examples

```
# read in a shapefile with demographic data
data(suffolk.map)

# choose number of districts
ndists <- 5

# create some initial plans
kplan <- createKmeansPlan(suffolk.map,ndists)
```

```

rplan <- createRandomPlan(suffolk.map,ndists)
aplan <- createAssignedPlan(suffolk.map,predvar=1:ndists)
wkplan<-createWeightedKmeansPlan(suffolk.map,ndists,weightVar="POP")

cplan <- createContiguousPlan(suffolk.map,ndists)
plot(cplan)
summary(cplan)

plot(kplan)
summary(kplan)
plot(summary(rplan))

c1plan<-createGreedyContiguousPlan(suffolk.map,ndists)

plot(c1plan)
summary(c1plan)

# single district only
c2plan<-createContiguousDistrict(suffolk.map,ndists)
c2plan
plot(c2plan)
summary(c2plan)

```

editPlanInteractive *Create and edit plans interactively, with a GUI*

Description

These functions allow one to create or edit a plan interactively, by selecting blocks from a map and assigning them to districts. Reports can be generated for each selection.

Usage

```

editPlanInteractive(plan, reportFUN = NULL, ...)
createPlanInteractive(basemap, ndists, reportFUN = NULL, ...)

```

Arguments

plan	plan to be edited
basemap	base map
ndists	number of districts
reportFUN	this function will be called after every selection is added, for continuous reporting of district scores
...	other arguments to pass to reportFUN

Value

returns a bard plan

warning

This depends on the iplots package. This package is still beta-quality. You may see occasional glitches. And on Macintosh systems, iplots must be run under JGR

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Micah Altman, 1997. "Is Automation the Answer? The Computational Complexity of Automated Redistricting", Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

Bush v. Vera, 517 U.S. 952 (1996).

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

C. Cirincione, T.A. Darling, and T.G. O'Rourke. 2000. "Assessing South Carolina's 1990's Congressional Districting." *Political Geography* 19: 189-211.

Grofman, B. 1982, "For single Member Districts Random is Not Equal", In *Representation and Redistricting Issues*, ed. B. Grofman, A. Lijphart, R. McKay, H. Scarrow. Lexington, MA: Lexington Books.

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. "From Crayons to Computers: The Evolution of Computer Use in Redistricting" *Social Science Computer Review* 23(3): 334-46.

See Also

Other plan creation functions [createRandomPlan](#), etc. Plan refinement algorithms [refineGreedyPlan](#), [refineAnnealPlan](#), [refineGenoudPlan](#), [refineNelderPlan](#)

Examples

```
# read in a shapefile with demographic data
data(suffolk.map)

# choose number of districts
ndists <- 5

# create some initial plans
kplan <- createKmeansPlan(suffolk.map,ndists)
```

```
## Not run:
if (require("iplots",quietly=TRUE)) {
  kplan<-editPlanInteractive(kplan, reportFUN=function(x)print(calcPopScore(x)))
}

## End(Not run)
```

fillHolesPlan	<i>Fixes up districting plan.</i>
---------------	-----------------------------------

Description

These functions assign unassigned blocks, and join non-contiguous regions.

Usage

```
fixUnassignedPlan(plan,method=c("random","fixed","closest"), fixed=1)
fillHolesPlan(plan,method=c("random","fixed","closest"), fixed=1)
fixContiguityPlan(plan,scoreFUN=NULL,...,display=FALSE)
```

Arguments

plan	input plan
method	Hole filling method to use
fixed	id for "fixed" method
scoreFUN	score to minimize when merging non-contiguous portions, if omitted, minimizes number of blocks reallocated
...	additional arguments to pass to score function
display	dynamic plotting for demos and teaching

Details

fillUnassignedPlan - Assigns non-assigns plocks . Fixed method assigns all missing blocks to a fixed value. Random method assigns blocks randomly. Closest assigns to a randomly chosen adjoining district (iteratively, if blocks are surrounded by other missing blocks). (fixHolesPlan is an alias for fillUnassignedPlan)

fixContiguityPlan - Evaluates all districts. For each disctict with noncontiguous portions, iteratively assigns to an existing contiguous district, greedily minimizing the given score function.

Value

Returns a bard plan.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Micah Altman, 1997. “Is Automation the Answer? The Computational Complexity of Automated Redistricting”, Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. “From Crayons to Computers: The Evolution of Computer Use in Redistricting” Social Science Computer Review 23(3): 334-46.

See Also

Plan generation algorithms: [createRandomPlan](#), [createKmeansPlan](#), [createContiguousPlan](#), [createRandomPopPlan](#), [createAssignedPlan](#).

Examples

```
data(suffolk.map)
kplan <- createKmeansPlan(suffolk.map,5)
kplan2<-kplan
is.na(kplan2[c(1,10,20,100)])<-TRUE
print(kplan2)
kplan3 <- fillHolesPlan(kplan2,method="closest")

# create non contiguous plan
rplan<-createRandomPopPlan(suffolk.map,5)

# fix it
rplanc<-fixContiguityPlan(rplan,calcPopScore)
```

hplot

HTML methods for BARD objects

Description

BARD provides various functions to make it easy to produce HTML output with R2HTML

Usage

```
hplot(x,y, ... , htmlFile=NULL, graphfileBase=NULL, bitmapArgs=NULL,htmlArgs=NULL)
copyR2HTMLfiles(outDir)
```

Arguments

x	object to plotted
y	optional object to plotted
...	additional parameters passed to plot.default
htmlFile	location of html file to write to, will default to contents of .HTML.file or tempdir/index.html

graphfileBase	base name for graph files , defaults to "graph"
bitmapArgs	arguments to pass to bitmap
htmlArgs	arguments to pass to HTMLInsertFile
outDir	target directory

Value

None

Note

hplot can be used instead of plot to easily insert plots into the HTML output. it calls plot() to plot the object, bitmap() to output it as bitmaps, and HTMLInsertGraph to generate HTML referring to it.

copyR2HTMLfiles is used to copy the css files and other support files needed by R2HTML into a temporary directory

HTML methods for BARD objects pretty prints those in HTML.

Author(s)

Micah Altman <micah_altman@harvard.edu>

See Also

[reportPlans](#), [HTMLInsertGraph](#), [bitmap](#), [plot](#)

Examples

```

data(suffolk.map)
numberdists <- 5
kplan <- createKmeansPlan(suffolk.map,numberdists)
rplan <- createRandomPlan(suffolk.map,numberdists)
if (require("R2HTML",quietly=TRUE)) {
targetDIR<- file.path(tempdir(),"R2HTML")
dir.create(targetDIR)
copyR2HTMLfiles(targetDIR)
target <- HTMLInitFile(targetDIR,filename="sample", BackgroundColor="#BBBBEE", Title="BARD Web Output")
HTML.title("BARD on the web", HR=2,file=target)
HTML("<p>Example of web output using BARD</p>",file=target)
hplot(kplan)
HTML("<p>Plan details</p>",file=target)
HTML(kplan,file=target)
HTMLEndFile()
## Not run:
browseURL(paste("file://",target,sep=""))

## End(Not run)
}

```

iacounty.sdf	<i>suffolk map</i>
--------------	--------------------

Description

county level map data

Usage

```
data(iacounty.sdf)
```

Format

Format is a SpatialPolygonsDataFrame

Details

A spatial data frame object of 2000 Iowa counties

Examples

```
data(iacounty.sdf)
```

importBardShape	<i>Import and export bard plans and basemaps as shapefiles</i>
-----------------	--

Description

These are convenience functions to read and write BARD data as ESRI shapefiles. For a faster method to store these in native form see readBardImage

Usage

```
exportBardShape(plan, filen, id = "BARDPlanID", gal = paste(filen, ".GAL", sep = ""))
importBardShape(filen, id="BARDPlanID", gal=paste(filen, ".GAL", sep=""), wantplan=FALSE, projection =
  as.character(NA), queen=TRUE, ...)
```

Arguments

filen	Name (and path to) file to be read or written
id	name of polygon block (NOT plan) id variable
plan	Plan assignment
wantplan	whether to extract embedded BARD plan from shapefile
projection	map projection

gal	GAL style contiguity list. Contiguity list will be regenerated if not supplied, but this is slow.
queen	Whether to use queen contiguity in poly2nb
...	additional arguments to pass to readShapePoly

Value

Read method returns a basemap by default. If "wantplan" is TRUE, read method returns a list with:

plan	a plan, as identified by the BARDplan variable in the shapefile
basemap	the basemap

Write method returns logical success, invisibly.

Note

- Using createAssignedPlan with an imported plan is generally more memory efficient than using wantplan=TRUE.

- Creating a contiguity list has been dramatically sped up ,however it can be somewhat slow for very large files (> 100000 units) unless rgeo is installed. Using GEODA or another program to generate GAL files is much faster for huge maps.

- A reasonable map projection should be used. If the map is unprojected, area calculations are potentially inaccurate. If rgdal is available BARD will attempt to reproject an unprojected map for increased accuracy.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Other methods for [readBardCheckpoint](#), [writeBardCheckpoint](#), [readBardImage](#), [writeBardImage](#), [readBardMap](#), [writeBardMap](#)

Examples

```
# read in a shapefile with demographic data
suffolk.map <- importBardShape(file.path(system.file("shapefiles", package="BARD"), "suffolk_tracts"))

# choose number of districts
ndists <- 5

# create some initial plans
kplan1 <- createKmeansPlan(suffolk.map, ndists)

# read and write images
exportBardShape(file.path(tempdir(), "shape1"), plan=kplan1)

# reimport
```

```
suffolk.map2<-importBardShape(file.path(tempdir(),"shape1"))
kplan2<-createAssignedPlan(suffolk.map2)
```

importBlockEquiv *Read and write block equivalency files*

Description

These are convenience functions to read block equivalency files.

Usage

```
importBlockEquiv(filen,basemap,idvar="GEOID10")
exportBlockEquiv(filen,plan,idvar="GEOID10")
blockEquiv2bardPlan(bedf,basemap,idvar="GEOID10")
```

Arguments

filen	Name (and path to) file to be read or written
basemap	single bard map
bedf	block equivalency data frame
plan	plan to write
idvar	name of variable containing block ids in bardMap

Value

Returns: bard plan

Note

A block equivalency file is a comma-separated values file with no column/row names, no quotes or comments, and either two or three columns.

The first column indicates the geographic identifier at the block level. The second column indicates the numeric district identifier. The third (optional) column indicates the number of seats in (magnitude of) the district.

For other file formats, use `read.table`, construct a 2 or 3 column data frame, and then use `blockEquiv2bardPlan`.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Other methods for [readBardCheckpoint](#), [writeBardCheckpoint](#), [importBardShape](#), [exportBardShape](#)

Examples

```
# read in a shapefile with demographic data
data(suffolk.map)
kplan<-createKmeansPlan(suffolk.map,5)

# read and write images
exportBlockEquiv(paste(tempdir(),"/be.csv",sep=""), kplan,idvar="TRACTID")
kplan2<-importBlockEquiv(paste(tempdir(),"/be.csv",sep=""),suffolk.map,idvar="TRACTID")
kplan3<-blockEquiv2bardPlan(data.frame(as.data.frame(suffolk.map)["TRACTID"],as.vector(kplan)),suffolk.map, id
```

miniball

compute minimum/bounding ball/circle

Description

Will compute the minimum ball (aka, bounding ball, bounding circle, minimum circle, spheroid hull) of a set of points.

Usage

```
miniball (points, pivot=TRUE, distances=FALSE)
```

Arguments

points	A n by r matrix of r -dimensional points.
pivot	Use pivoting methods for numerical stability
distances	Return vector of distances of points from center

Details

For those writing their own BARD score functions only. Returns the center and squared radius of the ball, support points, distances and tolerances as a list.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> (R interfaces)

http://www.hmdc.harvard.edu/micah_altman/ [R Interfaces] (Miniball C++ code by B. Gartner)

References

B. Gartner, 1999, "Fast and robust smallest enclosing balls", In *Proc. ESA*. '99

Examples

```
# simple example
xy <- matrix(runif(50),25)
plot(xy)
mb <- miniball(xy)

#distances from point to center
dc <- sqrt(rowSums(t(t(xy)-mb$center)^2))
```

 nb2graph

Neighborhood list helper functions

Description

These functions operate on neighborhood lists generated by the `spdep` package.

Usage

```
nb2graph(nb)
neighbors(nb,i,...)
neighbors.nb(nb,i,...)
n.comp.include(nb,include)
```

Arguments

<code>nb</code>	a neighborhood list generated by the <code>spdep</code> package
<code>i</code>	block id
<code>include</code>	exclusion list
<code>...</code>	additional arguments to pass onto neighbor methods

Details

These are primarily `spdep` internal functions, exposed for programmers.

`neighbors` is a generic method returning a list of indexes of all blocks directly connected to those in the supplied blocks id list.

`nb2graph` converts a *spdep* neighborhood object into a graph object suitable for use with *rbgl*

`n.comp.exclude` is analogous to `n.comp.nb` in that it returns the number of connected subgraphs induced by the neighborhood graph. However, it allows an exclusion list to be supplied, so that only a subset of the list is examined. This is semantically equivalent to `n.comp.nb(subset(nb,i))` but is roughly 10-50x faster.

Value

`nb2graph` returns a graph object, the others return a vector block ids

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

[n.comp.nb](#)

Examples

```
data(suffolk.map)
snb<-suffolk.map$nb
neighbors(snb,c(1,2,3))
kplan <- createKmeansPlan(suffolk.map,5)
rplan <- createRandomPlan(suffolk.map,5)
n.comp.include(snb,rplan==1)
n.comp.include(snb,kplan==1)
```

PMPreport

generate a specialized summary report on set of plans, for the Public Mapping Project

Description

This generates a districting report on district characteristics. It is used in the Public Mapping Project system.

Usage

```
PMPreport(bardMap,
blockAssignmentID="BARDPlanID",
popVar=NULL,
popVarExtra=NULL,
ratioVars=NULL,
splitVars = NULL,
blockLabelVar=NULL,
repCompactness=TRUE,
repCompactnessExtra=FALSE,
repSpatial=TRUE,
repSpatialExtra=FALSE,
useHTML=TRUE,
districtPolys=NULL,
...)
```

Arguments

<code>bardMap</code>	a bard basemap or plan.
<code>blockAssignmentID</code>	Not needed if <code>bardMap</code> is a plan. If <code>bardMap</code> is a basemap, name of variable in the basemap containing plan id's or a vector of ID assignments, as per <code>as.numeric(plan)</code> on any bard plan
<code>popVar</code>	List containing single population variable, and a tolerance variable for equal population. This will generate a population equality report . Use NULL to suppress this report.
<code>popVarExtra</code>	list of multiple additional demographic variables. The levels and district homogeneity of these will be reported. Use NULL to suppress this report
<code>ratioVars</code>	A nested list of lists. Each sub list contains a demonitator, numerator and threshold. The resulting report will show which districts are cumulatively above the threshold for each variable
<code>splitVars</code>	A list of variables in the basemap. These variables should indicate for each unit in the basemap the ID of some higher level geography to which the unit belongs (e.g. a county) The resulting report will show the number of times each district splits the higher-level geography.
<code>blockLabelVar</code>	The name of each underlying unit
<code>repCompactness</code>	Flag. Whether to include a report on compactness.
<code>repCompactnessExtra</code>	Flag. Include more compactness scores in the compactness report.
<code>repSpatial</code>	Flag. Report on unassigned blocks.
<code>repSpatialExtra</code>	Flag. Also report on contiguity, other spatial features.
<code>useHTML</code>	use html formatting for reports
<code>districtPolys</code>	aggregated polygons, one for each district, used for optimization
<code>...</code>	arguments passed on to <code>print</code> and <code>HTML</code>

Details

This produces a detailed report. See the examples.

Value

Nothing. The function is used for printing and plotting effect....

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

<http://publicmapping.org>

See Also

Scoring functions: [calcContiguityScore](#) Other reports comparing multiple plans: [reportPlans](#)

Examples

```

data(suffolk.map)

# choose number of districts
ndists <- 5

# create some initial plans
kplan <- createKmeansPlan(suffolk.map, ndists)
is.na(kplan[c(1,20,200)]) <- TRUE

# name the districts for the report...
levels(kplan) <- c("A", "Beta", "3+", "Fred's District", "Ceci n'est pas une district")

gpclibPermit()
aggregatePolys <- unionSpatialPolygons(bardBasemap2spatialDataFrame(suffolk.map), kplan)

PMPreport(kplan, popVar=list("Total Population"="POP", tolerance=.01), popVarExtra=list("Families"="FAMILIES", "V
"Majority Minority Districts"=list(
denominator=list("Total Population"="POP"),
threshold=.6,
numerators=list("Black Population"="BLACK", "Hispanic Population"="HISPANIC")
)
), splitVars = list("Tract"="TRT2000", "County"="FIPSSTCO"), blockLabelVar="ID",
repCompactness=TRUE, repCompactnessExtra=TRUE,
repSpatial=TRUE, repSpatialExtra=TRUE,
useHTML=FALSE, districtPolys=aggregatePolys)

#alternately -- can be a factor or integer list -- gaps don't matter
fplan <- factor(2*kplan, labels=levels(kplan))
PMPreport(suffolk.map, blockAssignmentID=fplan, popVar=list("Total Population"="POP", tolerance=.01), popVarExtra
"Majority Minority Districts"=list(
denominator=list("Total Population"="POP"),
threshold=.6,
numerators=list("Black Population"="BLACK", "Hispanic Population"="HISPANIC")
)
), splitVars = list("Tract"="TRT2000", "County"="FIPSSTCO"), blockLabelVar="ID",
repCompactness=TRUE, repCompactnessExtra=TRUE,
repSpatial=TRUE, repSpatialExtra=TRUE,
useHTML=FALSE, districtPolys=aggregatePolys)

```

Description

These functions creates sets of redistricting plans to show how redistricting criteria affect plan outcomes

Usage

```
profilePlans( seedplans, score.fun, ngenplans = 0,
  gen.fun = "createRandomPlan", gen.args = list(),
  refine.fun = "refineAnnealPlan", refine.args = list(),
  addscore.fun = NULL,
  weight.fun = function(score1, score2, weight)
    { sum(score1 + weight * score2)}, weight = seq(0, 1,
    length.out = 10, ), numevals = 10, tracelevel = 1,
  usecluster=TRUE)

samplePlans(seedplans, score.fun = calcContiguityScore,
  ngenplans = 24, gen.fun = "createRandomPlan", gen.args = list(),
  refine.fun = "refineAnnealPlan", refine.args = list(),
  tracelevel = 1,usecluster=TRUE)

quickSampleDistricts(ngenplans, basemap, ndists, distFUN=createContiguousDistrict, ...)
```

Arguments

seedplans	initial plans to be used as seeds
score.fun	base score function
ngenplans	number of additional plans to generate
gen.fun	function for generating additional plans
gen.args	a list of additional arguments to gen.fun
refine.fun	function for plan refinement
refine.args	a list of additional arguments to planRefineFun
addscore.fun	additional score component
weight.fun	function to generate weighted score
weight	vector of weight
numevals	number of evaluations per plan at each point
tracelevel	indicates desired level of printed tracing of optimization, 0 = no printing, higher levels give more detail. Nine is maximum
usecluster	use the bard cluster for computations if available
basemap	Bard basemap from which to create plan
ndists	number of districts to create
distFUN	district generating function to use
...	arguments to pass to distFUN

Details

samplePlans generates a set of plans, adds these to the seed plans given, and refines them based on the score function, to create a pseudo-sample of plans optimizing a particular score. profilePlans generates a set of plans pseudo-sampled using a two-part score function, where the weight of the second part is repeatedly.

quickSampleDistricts is a quick-and-dirty way of sampling a contiguous, equipopulous district (which may or may not be part of a plan containing only contiguous districts) These sample districts may be scored using the scoring functions, but may not be used with plan refinement functions at present.

Value

Returns a list of bard plans.

Note

samplePlans and profilePlans can be very compute intensive. If a compute cluster is configured, these functions will automatically distribute the computing load across the cluster. See startBardCluster.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

o Micah Altman, 1997. "Is Automation the Answer? The Computational Complexity of Automated Redistricting", Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

C. Cirincione , T.A. Darling, and T.G. O'Rourke. 2000. "Assessing South Carolina's 1990's Congressional Districting." *Political Geography* 19: 189-211.

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. "From Crayons to Computers: The Evolution of Computer Use in Redistricting" *Social Science Computer Review* 23(3): 334-46.

See Also

Plan refinement algorithms [refineGreedyPlan](#), [refineAnnealPlan](#), [refineGenoudPlan](#), [refineNelderPlan](#)

Cluster computing: [startBardCluster](#)

Examples

```

data(suffolk.map)

numberdists <- 5

# district sampling - quick

randomDists<-quickSampleDistricts(10,suffolk.map,numberdists)
distscores<- scorePlans(randomDists,scoreFUNs=list("LWCompact"=calcLWCompactScore,"PACompact"=calcPACompactScore))
plot(distscores[2:3])

# plan sampling and refinement -- compute intensive

kplan <- createKmeansPlan(suffolk.map,numberdists)
rplan <- createRandomPlan(suffolk.map,numberdists)
rplan2 <- createRandomPopPlan(suffolk.map,numberdists)

myScore<-function(plan,...) {
  return(calcContiguityScore(plan,...))
}

samples<-samplePlans(kplan, score.fun=myScore, ngenplans=20, gen.fun = "createRandomPlan", refine.fun="refineNearest")

profplans<-profilePlans( list(kplan,rplan), score.fun=calcContiguityScore, addscore.fun=calcPopScore, numevals=100)

summary(samples)
plot(summary(samples))
reportPlans(samples)
plot(summary(profplans))

```

readBardCheckpoint	<i>Write checkpoint of BARD state, read checkpoint and restart.</i>
--------------------	---

Description

This checkpoints the BARD program state as an R image, and can be used to restart BARD.

Usage

```

readBardCheckpoint(filen, continue = TRUE)
writeBardCheckpoint(filen, restart.fun = NULL)

```

Arguments

file	Name (and path to) file to be read or written
continue	Run restart function
restart.fun	Function to run after reading back checkpoint to restart

Details

These functions read and write parts of the .GlobalEnv.

Value

Write return logical success invisibly. Read returns logical success invisibly, and restores selected .GlobalEnv state. If doContinue is TRUE, and read is successful, it launches restart function on exit.

Note

readBardCheckpoint is not guaranteed to work with checkpoints saved in previous versions of BARD. Use [exportBardShape](#) in the previous version [importBardShape](#) to migrate.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Other methods for [readBardImage](#), [writeBardImage](#), [readBardMap](#), [writeBardMap](#), [exportBardShape](#), [importBardShape](#)

Examples

```
# read in a shapefile with demographic data
suffolk.map <- importBardShape(system.file("shapefiles/suffolk_tracts.shp", package="BARD"))

# choose number of districts
ndists <- 5

# create some initial plans
kplan1 <- createKmeansPlan(suffolk.map, ndists)
kplan2 <- createKmeansPlan(suffolk.map, ndists)

# read and write images
writeBardCheckpoint(paste(tempdir(), "/checkpoint1", sep=""),
  restart.fun=function(){cat("Welcome back\n")})
readBardCheckpoint(paste(tempdir(), "/checkpoint1", sep=""))
```

readBardImage	<i>Read and write BARD basemaps and plans as R images.</i>
---------------	--

Description

These are convenience functions to read and write BARD data as R images. This uses the native R image format, which is fast to read and to write. For a more portable, but slower approach see `importBardShape`

Usage

```
readBardImage(filename)
writeBardImage(filename, basemaps=NULL, plans=NULL)
```

Arguments

<code>filename</code>	Name (and path to) file to be read or written
<code>basemaps</code>	List of BARD basemaps to be written
<code>plans</code>	List of BARD plans to be written

Value

Read method returns:

<code>plans</code>	list of plans
<code>basemaps</code>	list of basemaps

Write method returns logical success, invisibly.

Note

- `readBardImage` attempts to detect and convert images created by previous versions of BARD, however it may fail to convert them due to limits in the underlying R libraries it depends. Use `exportBardShape` in the previous version `importBardShape` to migrate if automatic conversion is unsuccessful.

- Note that because of limitations in R load memory use grows when restoring multiple plans. The most memory efficient approach is to save the plans as vectors using `as.numeric`, save maps with `writeBardMap` and recreate the plan with `createAssignedPlan`.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Other methods for `readBardCheckpoint`, `writeBardCheckpoint`, `importBardShape`, `exportBardShape`

Examples

```
# read in a shapefile with demographic data
suffolk.map <- importBardShape(system.file("shapefiles/suffolk_tracts.shp", package="BARD"))

# choose number of districts
ndists <- 5

# create some initial plans
kplan1 <- createKmeansPlan(suffolk.map,ndists)
kplan2 <- createKmeansPlan(suffolk.map,ndists)

# read and write images
writeBardImage(paste(tempdir(),"/image1",sep=""),
  basemaps=list(suffolk.map),plans=list(kplan1,kplan2))
tmp.image<-readBardImage(paste(tempdir(),"/image1",sep=""))
```

readBardMap

Read and write single BARD basemaps fast.

Description

These are convenience functions to read and write BARD data as R images. This uses the native R image format, which is fast to read and to write. Specialized techniques avoid memory duplication.

Usage

```
readBardMap(filename,verbatim=FALSE)
writeBardMap(filename, basemap,verbatim=FALSE)
```

Arguments

filename	Name (and path to) file to be read or written
basemap	single bard map
verbatim	if false appends <code>_bard_nosave.Rdata</code> to filename <i>*if*</i> filename does not already contain this

Value

Read method returns:

basemap	BARD basemaps
---------	---------------

Write method returns logical success, invisibly.

Note

readBardImage attempts to detect and convert images created by previous versions of BARD, however it may fail to convert them due to limits in the underlying R libraries it depends. Use [exportBardShape](#) in the previous version [importBardShape](#) to migrate if automatic conversion is unsuccessful.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Other methods for [readBardCheckpoint](#), [writeBardCheckpoint](#), [importBardShape](#), [exportBardShape](#)

Examples

```
# read in a shapefile with demographic data
suffolk.map <- importBardShape(system.file("shapefiles/suffolk_tracts.shp", package="BARD"))

# read and write images
writeBardMap(paste(tempdir(),"/image1",sep=""), basemap=suffolk.map)
tmp.image<-readBardMap(paste(tempdir(),"/image1",sep=""))
```

refineGreedyPlan

Create an initial plan for later refinement

Description

These methods create plans starting with random seeds. These plans should be refined with one of the refinement functions.

Usage

```
refineGreedyPlan(plan, score.fun, displaycount=NULL, historysize=0, dynamicscoring=FALSE, tracelevel=1)
refineGenoudPlan(plan, score.fun, displaycount = NULL, historysize = 0,
                 dynamicscoring = FALSE, usecluster = TRUE, tracelevel=1)
refineAnnealPlan(plan, score.fun, displaycount = NULL, historysize = 0,
                 dynamicscoring = FALSE, tracelevel=1, checkpointCount=0, resume=FALSE, greedyFinish=FALSE)
refineNelderPlan( plan, score.fun, displaycount = NULL, maxit = NULL,
                 historysize = 0, dynamicscoring = FALSE, tracelevel = 1)
refineTabuPlan( plan, score.fun, displaycount = NULL, historysize = 0,
                 dynamicscoring = FALSE, tracelevel = 1, tabusize = 100, tabusample=500, checkpointCount=0, r
refineGRASPPlan (plan, score.fun, samplesize = 50, predvar = NULL,
                 displaycount = NULL, historysize = 0,
                 dynamicscoring = FALSE, usecluster = TRUE, tracelevel = 1)
```

Arguments

plan	plan to be refined, this can be randomly generated through the plan generation functions below, or can be an existing real plan
score.fun	a function that accepts a plan and returns a single score, or a vector of scores, which will be summed to produce the overall plan score. The score function should accept the arguments, plan, lastscore, and changelist. The last two are for incremental scores, and their values may be ignored if the score function does not support incremental score recalculation. See calcPopScore for an example of a score function.
usecluster	whether to use the defined bard computing cluster, if available. Only refineGenoudPlan, and refineGRASPPlan implements cluster level parallelism at this time. Support for rgenoud clustering with BARD is experimental – performance is likely to be very bad unless the connections are very fast, and only snow clusters are supported.
displaycount	This is primarily for demos. It updates a plot of the last plan to be scored (which may not be the current iteration’s best scoring candidate) every N iterations
tracelevel	Provides a trace of the algorithms progress. For debugging. Currently the legal values are 0 (silence), 1 (minimal information), 2 (more information), 3 (maximum information)
dynamicscoring	Makes use of incremental score functions by tracking changes between candidate plans and scoring only the incremental changes. Not implemented for refineGenoudPlan, since multiple candidates are evaluated simultaneously. Use this only if score functions are expensive to compute.
historysize	Keep a score history of size n, in order to avoid reevaluating scores for plans already seen. Not currently useful for refineGreedyPlan, since it never revisits plans. Plan digests are used to avoid storing many full plans in memory, however this increases the computation necessary to track the history. Use this when computing a score function is very expensive.
greedyFinish	Refine final solution with hill-climbing
maxit	Maximum number of iterations before stopping
predvar	Population variable for GRASP plan
samplesize	Number of samples to use for GRASP
tabusize	Size of tabu list
tabusample	Number of samples to take at each iteration
checkpointCount	Saves checkpoints to global BardCheckPoint variable, of internal state every Nth iteration, so that function can be resumed
resume	Resume interrupted optimization, from checkpointed state
doquench	Fast quenching – for a greedy solution
doReanneal	Attempt to reanneal when no progress is made – for a more effective optimization
...	parameters to send to interenal reAnneal command for debugging

Details

Greedy – The greedy algorithm uses hill climbing: at every iteration it compares all possible assignments of single blocks to a spatially contiguous district, and selects the best scoring. Stops when no improvement can be made. This can be interrupted and restarted with the checkpoint and resume arguments.

Tabu – A modification of the greedy method. Samples the assignments of simple blocks. Always takes a move that yields a better solution than the best seen before. Otherwise takes a trade that yields something better than the most recent seen, unless that trade has been used recently and is still on the tabu list. This can be interrupted and restarted with the checkpoint and resume arguments.

Genoud – uses the [genoud](#) implementation of the genetic algorithm. This function tunes the nine genoud operators for the redistricting problem.

Anneal – uses simulated annealing. A plan generation function is provided that generates plans through one-way assignments or two-way exchanges between neighboring blocks of different districts.

Nelder – uses nelder-mead. For demonstration only. Not very effective.

GRASP – GRASP uses greedy randomized adaptive search. Essentially this randomly generates plans via randomPopPlan, and refines using greedy refinement. Greedy refinement is also used on the seed plan. The best of the set is returned. Extremely compute intensive and will make use of clusters if available.

Value

Returns a bard plan.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Micah Altman, 1997. “Is Automation the Answer? The Computational Complexity of Automated Redistricting”, Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. “From Crayons to Computers: The Evolution of Computer Use in Redistricting” *Social Science Computer Review* 23(3): 334-46.

See Also

Plan generation algorithms: [createRandomPlan](#), [createKmeansPlan](#), [createContiguousPlan](#), [createRandomPopPlan](#), [createAssignedPlan](#).

Scoring functions: [calcContiguityScore](#)

Examples

```

data(suffolk.map)
numberdists <- 5
rplan <- createRandomPlan(suffolk.map,numberdists)
myScore<-function(plan,...){calcContiguityScore(plan,standardize=FALSE)}
# for example - not very effective
improvedRplan<-refineNelderPlan(plan=rplan,score.fun=myScore,displaycount=50,tracelevel=0,maxit=300)
## Not run:
improvedRplan<-refineTabuPlan(plan=rplan,score.fun=myScore)

## End(Not run)

```

reportPlans

evaluate a set of plans

Description

This function evaluates scores, and differences among a set of plans.

Usage

```

reportPlans(
  plans,
  scoreFUNs=list(
    "Contiguity"=calcContiguityScore,
    "Holes"=calcHolesScore,
    "LW Compact"=calcLWCompactScore,
    "Reock"=calcReockScore
  ),
  doplots=FALSE,
  domatch=TRUE,
  dodiff=TRUE,
  dodetails=FALSE,
  doprofileextras=TRUE,
  plotOpts=NULL,
  useHTML=FALSE,
  completeHTML=FALSE,
  ...
)

```

Arguments

plans	a list of bard plans.
scoreFUNs	a list of named score functions that accept a plan as an argument, and return a vector of scores

domatch	logical, whether to attempt to reorder district ID's to match
doplot	Logical. Whether to plot differences.
dodetails	Logical. Print detailed information
dodiff	Logical. Report differences between pairs of plans
doprofileextras	Logical. Report profile summaries for bardSample results
plotOpts	List of plotting options to send to plan plotting command
useHTML	use html formatting for reports
completeHTML	produce a complete HTML file – note the path is returned
...	arguments passed on to print and HTML

Details

This is the externally visible routine for comparing a list of plans. If multiple plans are given, each is compared against the first plan in the list.

Value

Nothing. The function is used for printing and plotting effect.... unless the completeHTML option is used, in which case returns the path the the HTML file, invisibly.

Note

Note the following limitation: all plans being compared must have the same number of districts and basemap

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

References

Micah Altman, 1997. “Is Automation the Answer? The Computational Complexity of Automated Redistricting”, Rutgers Computer and Technology Law Journal 23 (1), 81-142 http://www.hmdc.harvard.edu/micah_altman/pubpapers.shtml

Altman, M. 1998. Modeling the Effect of Mandatory District Compactness on Partisan Gerrymanders, *Political Geography* 17:989-1012.

Micah Altman and Michael P. McDonald. 2004. A Computation Intensive Method for Detecting Gerrymanders Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois, Apr 15, 2004. http://www.allacademic.com/meta/p83108_index.html

Micah Altman, Karin Mac Donald, and Michael P. McDonald, 2005. “From Crayons to Computers: The Evolution of Computer Use in Redistricting” *Social Science Computer Review* 23(3): 334-46.

See Also

Scoring functions: [calcContiguityScore](#) Component functions: [scorePlans](#) , [diff.bardPlan](#)

Examples

```

data(suffolk.map)

# choose number of districts
ndists <- 5

# create some initial plans
kplan1 <- createKmeansPlan(suffolk.map,ndists)
kplan2 <- createKmeansPlan(suffolk.map,ndists)
reportPlans(plans=list("kmeans"=kplan1,"kmeans 2"=kplan2), doplot=TRUE)

```

scorePlans

*Methods for comparing plans***Description**

Compares plans by differences and such

Usage

```

scorePlans(plans, scoreFUNs, domatch = TRUE)
## S3 method for class 'bardPlan'
diff(x,plan2,domatch=TRUE,...)

```

Arguments

plans	list of plans
x	first plan
plan2	second plan
scoreFUNs	list of score functions
domatch	rearrange district ID's for a best match between two plans
...	ignored

Value

Score plans returns a score data frame diff returns a difference lists

Note

Use summary, and plot(summary) to display results

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also[reportPlans](#)**Examples**

```

data(suffolk.map)
numberdists <- 5
kplan <- createKmeansPlan(suffolk.map,numberdists)
rplan <- createRandomPlan(suffolk.map,numberdists)
pdiff <- diff(kplan,rplan)
# detailed
print(pdiff)
# numbers of changes
print(summary(pdiff))
# shows changed blocks on a map
plot(pdiff,plotall=TRUE)
# show scores
scorePlans(list("kmeans"=kplan,"random"=rplan),
scoreFUNs=list( "Contiguity"=calcContiguityScore,
                "Holes"=calcHolesScore,
                "LW Compact"=calcLWCompactScore,
                "Reock"=calcReockScore) )

```

spatialDataFrame2bardBasemap

conversion methods for BARD objects

Description

converts BARD basemap to SpatialPolygonsDataFrame and back

Usage

```

spatialDataFrame2bardBasemap(sdf,nb=NULL,queen=TRUE,keepgeom=TRUE)
spatialDataFrame2bardPlan(sdf,nb=NULL,queen=TRUE,id="BARDPlanID")
bardBasemap2spatialDataFrame(x)
bardPlan2spatialDataFrame(x,id="BARDPlanID")

```

Arguments

sdf	spatial data frame object
x	bardBasemap object
nb	object of class nb
queen	use queen contiguity when running poly2nb
id	name of column used to identify plan assingment
keepgeom	keep geometry

Value

returns basemap, plan, or spatialPolygonsDataFrame, respectively

Note

Conversion from Spatial Data Frame's to bard objects is potentially slow, because BARD objects use index structures not available in the sdf's, and the conversion must create these indices.

keepgeom=FALSE is only for use with a limited set of scoring functions, does not support plots or district generation. Avoid if you're not absolutely sure.

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

Examples

```
xx <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1], IDvar="FIPSNO", proj4string=CRS("+pr
data(suffolk.map)
suffolk.sdf <- bardBasemap2spatialDataFrame(suffolk.map)
xx.map<-spatialDataFrame2bardBasemap(xx, queen=TRUE)
data(iacounty.sdf)
iacounty.map<-spatialDataFrame2bardBasemap(iacounty.sdf, queen=FALSE)
rpplan<-createRandomPopPlan(iacounty.map, 10)
iacounty2.sdf <-bardPlan2spatialDataFrame(rpplan)
rpplan2<-spatialDataFrame2bardPlan(iacounty2.sdf)
```

startBardCluster

Use bard with snow distributed computing clusters.

Description

These functions configure a snow computing cluster to be used with bard profiling, sampling, and plan refinement

Usage

```
startBardCluster(c1=0)
stopBardCluster()
```

Arguments

c1 either 0, which attempts to use all cores, a specific number of cores to use, a cluster returned from snow, or a vector of cluster systems names

Details

This function attempts to configure a computing cluster for bard. If given a single 0, multicore will be used with all cores, if a single number, a fixed number of cores will be used. Otherwise if given an existing snow cluster, it will use that, or if given a vector of machine names it will use snows makeCluster to start a socket-based cluster. (You will be required to enter your login password for these systems, if you have not set up an existing ssh key.)

Plan sampling and profiling, and some plan refinement will automatically use a cluster that has been initialized.

stopBardCluster stops and disbands the cluster configured through startBardCluster. It is normally called automatically when the BARD module is unloaded.

Value

Returns a logical value indicating successful initialization.

Note

Initialization will fail if attempts to connect to the machines fail, or if BARD cannot be installed and started on these systems (an attempt to install BARD will automatically be made, if BARD is not installed.)

Author(s)

Micah Altman <Micah_Altman@harvard.edu> http://www.hmdc.harvard.edu/micah_altman/

See Also

Plan refinement algorithms [refineGenoudPlan](#)

Plan sampling: [samplePlans](#), [profilePlans](#)

Cluster computing: [makeCluster](#)

Examples

```
## Not run:
suffolk.map <- importBardShape(file.path(system.file("shapefiles", package="BARD"),"suffolk_tracts"))
numberdists <- 5
kplan <- createKmeansPlan(suffolk.map,numberdists)
rplan <- createRandomPlan(suffolk.map,numberdists)
myScore<-function(plan,...) {
  return(calcContiguityScore(plan,...))
}

# here is where we try to start the cluster!
# snow cluster
# startBardCluster(c("localhost","localhost"))
# multicore cluster
startBardCluster()

# this will use the cluster automatically
```

```
samples<-samplePlans(kplan, score.fun=myScore, ngenplans=4, gen.fun = "createRandomPlan", refine.fun="refineNeld
profplans<-profilePlans( list(kplan,rplan), score.fun=calcContiguityScore, addscore.fun=calcPopScore, numevals=

## End(Not run)
```

suffolk.map

suffolk map

Description

tract level map data

Usage

```
data(suffolk.map)
```

Format

The format is: List of 12 \$ shape :Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots \$ nb :List \$ df :'data.frame' \$ perims : num \$ sharedPerims:List \$ areas : num \$ bboxs : num \$ perims : num \$ centroids : num \$ timestamp : POSIXct[1:1] \$ longlat : logi - attr(*, "class")= chr "bardBasemap"

Details

Map of census tracts in Suffolk county, new york supplemented with 2000 demographic data.

Examples

```
data(suffolk.map)
```

Index

*Topic **IO**

- BARD-package, 2
- basem<- .default, 5
- calcLWCompactScore, 6
- choroplexPlan, 10
- combineDynamicScores, 11
- createRandomPlan, 13
- editPlanInteractive, 17
- hplot, 20
- importBardShape, 22
- importBlockEquiv, 24
- nb2graph, 26
- PMPreport, 27
- profilePlans, 29
- readBardCheckpoint, 32
- readBardImage, 34
- readBardMap, 35
- refineGreedyPlan, 36
- reportPlans, 39
- spatialDataFrame2bardBasemap, 42
- startBardCluster, 43

*Topic **cluster**

- miniball, 25

*Topic **datasets**

- iacounty.sdf, 22
- suffolk.map, 45

*Topic **distribution**

- BARD-package, 2
- calcLWCompactScore, 6
- choroplexPlan, 10
- combineDynamicScores, 11
- createRandomPlan, 13
- editPlanInteractive, 17
- PMPreport, 27
- refineGreedyPlan, 36
- reportPlans, 39
- scorePlans, 41

*Topic **iplot**

- hplot, 20

*Topic **models**

- BARD-package, 2
- calcLWCompactScore, 6
- choroplexPlan, 10
- combineDynamicScores, 11
- createRandomPlan, 13
- editPlanInteractive, 17
- PMPreport, 27
- refineGreedyPlan, 36
- reportPlans, 39
- scorePlans, 41

*Topic **optimize**

- BARD-package, 2
- calcLWCompactScore, 6
- choroplexPlan, 10
- combineDynamicScores, 11
- createRandomPlan, 13
- editPlanInteractive, 17
- PMPreport, 27
- refineGreedyPlan, 36
- reportPlans, 39

*Topic **print**

- hplot, 20

*Topic **spatial**

- BARD-package, 2
- basem<- .default, 5
- calcLWCompactScore, 6
- choroplexPlan, 10
- combineDynamicScores, 11
- createRandomPlan, 13
- editPlanInteractive, 17
- fillHolesPlan, 19
- importBardShape, 22
- importBlockEquiv, 24
- miniball, 25
- nb2graph, 26
- PMPreport, 27
- profilePlans, 29
- readBardCheckpoint, 32

- readBardImage, 34
- readBardMap, 35
- refineGreedyPlan, 36
- reportPlans, 39
- scorePlans, 41
- spatialDataFrame2bardBasemap, 42
- startBardCluster, 43

- BARD (BARD-package), 2
- BARD-package, 2
- bardBasemap2spatialDataFrame
 - (spatialDataFrame2bardBasemap), 42
- bardPlan2spatialDataFrame
 - (spatialDataFrame2bardBasemap), 42
- basem (basem<- .default), 5
- basem<- (basem<- .default), 5
- basem<- .default, 5
- bitmap, 21
- blockEquiv2bardPlan (importBlockEquiv), 24

- calcBBCompactScore, 3
- calcBBCompactScore
 - (calcLWCompactScore), 6
- calcContiguityScore, 3, 11, 29, 38, 40
- calcContiguityScore
 - (calcLWCompactScore), 6
- calcGroupScore, 3
- calcGroupScore (calcLWCompactScore), 6
- calcHolesScore (calcLWCompactScore), 6
- calcIneqScore, 3
- calcIneqScore (calcLWCompactScore), 6
- calcLWCompactScore, 3, 6
- calcMomentScore, 3
- calcMomentScore (calcLWCompactScore), 6
- calcPACompactScore, 3
- calcPACompactScore
 - (calcLWCompactScore), 6
- calcPopScore, 3
- calcPopScore (calcLWCompactScore), 6
- calcRangeScore, 3
- calcRangeScore (calcLWCompactScore), 6
- calcReockScore, 3
- calcReockScore (calcLWCompactScore), 6
- calcSpatialHolesScore, 3
- calcSpatialHolesScore
 - (calcLWCompactScore), 6
- calcSplitScore, 3
- calcSplitScore (calcLWCompactScore), 6
- calcUnassignedScore, 3
- calcUnassignedScore
 - (calcLWCompactScore), 6
- choroplexPlan, 3, 10
- combineDynamicScores, 3, 9, 11
- copyR2HTMLfiles (hplot), 20
- createAssignedPlan, 20, 38
- createAssignedPlan (createRandomPlan), 13
- createContiguousDistrict, 3
- createContiguousDistrict
 - (createRandomPlan), 13
- createContiguousPlan, 2, 20, 38
- createContiguousPlan
 - (createRandomPlan), 13
- createGreedyContiguousPlan, 2
- createGreedyContiguousPlan
 - (createRandomPlan), 13
- createKmeansPlan, 2, 20, 38
- createKmeansPlan (createRandomPlan), 13
- createPlanInteractive, 3
- createPlanInteractive
 - (editPlanInteractive), 17
- createRandomPlan, 2, 13, 18, 20, 38
- createRandomPopPlan, 2, 20, 38
- createRandomPopPlan (createRandomPlan), 13
- createWeightedKmeansPlan, 2
- createWeightedKmeansPlan
 - (createRandomPlan), 13

- diff.bardPlan, 3, 40
- diff.bardPlan (scorePlans), 41

- editPlanInteractive, 3, 16, 17
- exportBardShape, 2, 24, 33, 34, 36
- exportBardShape (importBardShape), 22
- exportBlockEquiv (importBlockEquiv), 24

- fillHolesPlan, 19
- fixContiguityPlan, 3, 16
- fixContiguityPlan (fillHolesPlan), 19
- fixUnassignedPlan, 3, 16
- fixUnassignedPlan (fillHolesPlan), 19

- genoud, 3, 38

- hplot, 20

HTMLInsertGraph, 21
 iacounty.sdf, 22
 importBardShape, 22, 24, 33, 34, 36
 importBlockEquiv, 2, 24
 ineq, 9

 makeCluster, 44
 miniball, 25

 n.comp.include (nb2graph), 26
 n.comp.nb, 27
 nb2graph, 26
 neighbors (nb2graph), 26

 plot, 21
 PMPreport, 3, 27
 profilePlans, 3, 29, 44

 quickSampleDistricts, 3
 quickSampleDistricts (profilePlans), 29

 readBardCheckpoint, 2, 23, 24, 32, 34, 36
 readBardImage, 23, 33, 34
 readBardMap, 2, 23, 33, 35
 readShapePoly, 3
 refineAnnealPlan, 3, 9, 13, 16, 18, 31
 refineAnnealPlan (refineGreedyPlan), 36
 refineGenoudPlan, 3, 9, 13, 16, 18, 31, 44
 refineGenoudPlan (refineGreedyPlan), 36
 refineGRASPPlan, 3
 refineGRASPPlan (refineGreedyPlan), 36
 refineGreedyPlan, 3, 9, 13, 16, 18, 31, 36
 refineNelderPlan, 3, 9, 13, 16, 18, 31
 refineNelderPlan (refineGreedyPlan), 36
 refineTabuPlan, 3, 9, 13
 refineTabuPlan (refineGreedyPlan), 36
 reportPlans, 3, 21, 39, 42

 samplePlans, 3, 44
 samplePlans (profilePlans), 29
 scorePlans, 3, 40, 41
 spatialDataFrame2bardBasemap, 42
 spatialDataFrame2bardPlan
 (spatialDataFrame2bardBasemap),
 42
 spdep, 3
 startBardCluster, 3, 31, 43
 stopBardCluster (startBardCluster), 43
 suffolk.map, 45

 writeBardCheckpoint, 23, 24, 34, 36
 writeBardCheckpoint
 (readBardCheckpoint), 32
 writeBardImage, 23, 33
 writeBardImage (readBardImage), 34
 writeBardMap, 2, 23, 33
 writeBardMap (readBardMap), 35