

Package ‘BARD’

October 30, 2009

Type Package

Title Better Automated ReDistricting

Version 1.05

Date 2009-10-29

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 digest

Imports spdep,iplots

Suggests snow,rgenoud,rJava

License AGPL-3 + file LICENSE

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

Repository CRAN

Date/Publication 2009-10-30 07:46:51

R topics documented:

BARD-package	2
basem<-.default	4
calcLWCompactScore	5
combineDynamicScores	8
createRandomPlan	10
editPlanInteractive	13
fillHolesPlan	15
importBardShape	16
miniball	18

nb2graph	19
profilePlans	20
readBardCheckpoint	22
readBardImage	23
refineGreedyPlan	24
reportPlans	27
scorePlans	29
startBardCluster	30

Index	33
--------------	-----------

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:	1.01
Date:	2009-1-2
License:	AGPL 3.0

Bard supports several areas of functionality: 1. Plan input output. Reading and writing plans in various formats. ([readBardImage](#), [writeBardImage](#), [readBardCheckpoint](#), [writeBardCheckpoint](#), [importBardShape](#), [exportBardShape](#))

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

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

4. Plan scoring. Scoring functions for use in plan refinement, profiling, and comparison. ([calcContiguityScore](#), [calcReockScore](#), [calcLWCompactScore](#), [calcGroupScore](#), [calcPopScore](#), [calcHolesScore](#), [calcRangeScore](#), [calcMomentScore](#))

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

6. Plan 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

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

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

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](#), [maptools](#), [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)

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

#just for quick demonstration -- nelder method not effective

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

## Not run:
# This works better, but will take a while
improvedRplan<-refineAnnealPlan(plan=rplan2, score.fun=myScore, historysize=0, dynamicscori=0.1)
## End(Not run)

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

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

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

```

basem<-.default *manipulate plan basemaps*

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
suffolk.map <- importBardShape(file.path(system.file("shapefiles", package="BARD"),"suffolk.shp"))
kplan<-createRandomPlan(suffolk.map,5)
copy.of.suffolk.map<-basem(kplan)
```

calcLWCompactScore *Scoring functions for redistricting.*

Description

These functions evaluate redistricting plans.

Usage

```
calcRangeScore(plan, predvar="BLACK", predvar2="WHITE", targrange=c(.65,.70),
  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 )
calcLWCompactScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
calcReockScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
calcContiguityScore(plan, lastscore=NULL, changelist=NULL, standardize=TRUE )
calcHolesScore(plan, lastscore = NULL, changelist = NULL, standardize = TRUE)
calcMomentScore(plan,standardize=TRUE,centers=NULL,weightVar=NULL,penaltyExp=2, lastscore=NULL)
```

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

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.

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 $\text{sum}(\text{prevar1})/(\text{sum}(\text{predvar1})+\text{sum}(\text{predvar2}))$ 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).

calcHolesScores - Penalizes plans for having unassigned blocks

calcMomentScore - calculates moment of inertia compactness score of the district, the sum or squared distance of the block centroids (weighted by are 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)

Value

All current plan score functions return a vector of score value, representing the score for each district, with the plan score being the sum of this vector. (User-written score functions MAY return a single number as a plan score instead, all bard utilities will handle this case correctly.)

If the 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 scored be used except for testing.

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

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](#).

Examples

```
# read in a shapefile with demographic data
suffolk.map <- importBardShape(
  system.file("shapefiles/suffolk_tracts.shp", package="BARD"))
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(kplan)
calcLWCompactScore(rplan)
calcLWCompactScore(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
cl<-cbind(c(318, 320), c(kplan1[318], kplan1[320]))

if ( ! all(calcPopScore(kplan2, lastscore=calcPopScore(kplan1), changelist=cl) == calcPopScore(kplan2, lastscore=calcPopScore(kplan1), changelist=cl)) )
  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")

```

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, scorefun=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
scorefuncs	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
suffolk.map <- importBardShape(
  system.file("shapefiles/suffolk_tracts.shp", package="BARD"))
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) == calcP
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!")
}

```

createRandomPlan *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

```

createRandomPlan(basemap, ndists)
createRandomPopPlan(basemap, ndists, predvar="POP")
createAssignedPlan(basemap, predvar="BARDPlanID")
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)

```

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

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]

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)

`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.

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](#)

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

## Not run:
cplan <- createContiguousPlan(suffolk.map, ndists)

## End(Not run)

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

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	bard basemap

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.

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

# 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>Fill holes in a redistricting plan.</i>
---------------	--

Description

This fills holes (unassigned blocks) in a redistricting plan, using selectable methods.

Usage

```
fillHolesPlan(plan, method=c("random", "fixed", "closest"), fixed=1)
```

Arguments

plan	input plan
method	Hole filling method to use
fixed	id for “fixed” method

Details

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)

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

```
suffolk.map <- importBardShape(
  file.path(system.file("shapefiles", package="BARD"), "suffolk_tracts")
)
kplan <- createKmeansPlan(suffolk.map, 5)
kplan2 <- kplan
is.na(kplan2[c(1, 10, 20, 100)]) <- TRUE
print(kplan2)
kplan3 <- fillHolesPlan(kplan2, method="closest")
```

importBardShape *Import and export bard plans and basemaps as shapefiles*

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)
```

Arguments

file	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
gal	GAL style contiguity list. Contiguity list will be regenerated if not supplied, but this is slow.

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`.

Author(s)

Micah Altman (Micah_Altman@harvard.edu) http://www.hmdc.harvard.edu/micah_altman/

See Also

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

Examples

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

# 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)
```

```
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

<code>points</code>	A n by r matrix of r -dimensional points.
<code>pivot</code>	Use pivoting methods for numerical stability
<code>distances</code>	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))
```

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

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

profilePlans

Profile or pseudo-sample redistricting plans based on score criteria.

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)
```

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

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.

Value

Returns a list of bard plans.

Note

These functions 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

```

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)

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

samples<-samplePlans(kplan, score.fun=myScore, ngenplans=20, gen.fun = "createRandomPlan",
profplans<-profilePlans( list(kplan,rplan), score.fun=calcContiguityScore, addscore.fun=cal

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

```

readBardCheckpoint *Write checkpoint of BARD state, read checkpoint and restart.*

Description

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

Usage

```

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

```

Arguments

filename	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.

Author(s)

Micah Altman (Micah_Altman@harvard.edu) http://www.hmdc.harvard.edu/micah_altman/

See Also

Other methods for [readBardImage](#), [writeBardImage](#), [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

Read and write BARD basemaps and plans as R images.

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(filen)
writeBardImage(filen, basemaps=NULL, plans=NULL)
```

Arguments

filen	Name (and path to) file to be read or written
basemaps	List of BARD basemaps to be written
plans	List of BARD plans to be written

Value

Read method returns:

plans	list of plans
basemaps	list of basemaps

Write method returns logical success, invisibly.

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=""))
```

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=
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, 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=
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 <code>calcPopScore</code> for an example of a score function.
usecluster	whether to use the defined bard computing cluster, if available. Only <code>refineGenoudPlan</code> , and <code>refineGRASPPlan</code> implements cluster level paralellism at this time. Support for <code>rgenoud</code> clustering with BARD is experimental – performance is likely to be very bad unless the connections are very fast.
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 <code>refineGenoudPlan</code> , since multiple candidates are evaluated simulatneously. 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 <code>refineGreedyPlan</code> , 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

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

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.

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

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

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

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.

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

```
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)
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

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

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(cl)
stopBardCluster()
```

Arguments

cl either 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 an existing snow cluster, it will use that. If given a vector of machine names it will use `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"), "
    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!
  startBardCluster(c("localhost","localhost"))

# this will use the cluster automagically
samples<-samplePlans(kplan, score.fun=myScore, ngenplans=4, gen.fun = "createRandomPlan", re
profplans<-profilePlans( list(kplan,rplan), score.fun=calcContiguityScore, addscore.fun=cal
```

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

Index

*Topic **IO**

- BARD-package, 2
- basem<- .default, 5
- calcLWCompactScore, 6
- combineDynamicScores, 9
- createRandomPlan, 11
- editPlanInteractive, 14
- importBardShape, 17
- nb2graph, 19
- profilePlans, 21
- readBardCheckpoint, 23
- readBardImage, 24
- refineGreedyPlan, 25
- reportPlans, 28
- startBardCluster, 31

*Topic **cluster**

- miniball, 18

*Topic **distribution**

- BARD-package, 2
- calcLWCompactScore, 6
- combineDynamicScores, 9
- createRandomPlan, 11
- editPlanInteractive, 14
- refineGreedyPlan, 25
- reportPlans, 28
- scorePlans, 30

*Topic **models**

- BARD-package, 2
- calcLWCompactScore, 6
- combineDynamicScores, 9
- createRandomPlan, 11
- editPlanInteractive, 14
- refineGreedyPlan, 25
- reportPlans, 28
- scorePlans, 30

*Topic **optimize**

- BARD-package, 2
- calcLWCompactScore, 6
- combineDynamicScores, 9

- createRandomPlan, 11
- editPlanInteractive, 14
- refineGreedyPlan, 25
- reportPlans, 28

*Topic **spatial**

- BARD-package, 2
- basem<- .default, 5
- calcLWCompactScore, 6
- combineDynamicScores, 9
- createRandomPlan, 11
- editPlanInteractive, 14
- fillHolesPlan, 16
- importBardShape, 17
- miniball, 18
- nb2graph, 19
- profilePlans, 21
- readBardCheckpoint, 23
- readBardImage, 24
- refineGreedyPlan, 25
- reportPlans, 28
- scorePlans, 30
- startBardCluster, 31

BARD (*BARD-package*), 2

BARD-package, 2

basem (*basem<- .default*), 5

basem<- (*basem<- .default*), 5

basem<- .default, 5

calcContiguityScore, 3, 27, 29

calcContiguityScore
(*calcLWCompactScore*), 6

calcGroupScore, 3

calcGroupScore
(*calcLWCompactScore*), 6

calcHolesScore, 3

calcHolesScore
(*calcLWCompactScore*), 6

calcLWCompactScore, 3, 6

calcMomentScore, 3

calcMomentScore
 (*calcLWCompactScore*), 6
 calcPopScore, 3
 calcPopScore
 (*calcLWCompactScore*), 6
 calcRangeScore, 3
 calcRangeScore
 (*calcLWCompactScore*), 6
 calcReockScore, 3
 calcReockScore
 (*calcLWCompactScore*), 6
 combinedynamicScores, 8, 9
 createAssignedPlan, 17, 27
 createAssignedPlan
 (*createRandomPlan*), 11
 createContiguousPlan, 3, 17, 27
 createContiguousPlan
 (*createRandomPlan*), 11
 createKmeansPlan, 3, 17, 27
 createKmeansPlan
 (*createRandomPlan*), 11
 createPlanInteractive
 (*editPlanInteractive*), 14
 createRandomPlan, 3, 11, 15, 17, 27
 createRandomPopPlan, 3, 17, 27
 createRandomPopPlan
 (*createRandomPlan*), 11
 createWeightedContiguousPlan
 (*createRandomPlan*), 11
 createWeightedKmeansPlan, 3
 createWeightedKmeansPlan
 (*createRandomPlan*), 11

 diff.bardPlan, 3, 29
 diff.bardPlan (*scorePlans*), 30

 editPlanInteractive, 3, 13, 14
 exportBardShape, 3, 24, 25
 exportBardShape
 (*importBardShape*), 17

 fillHolesPlan, 3, 16

 genoud, 4, 27

 importBardShape, 3, 17, 24, 25

 makeCluster, 32
 maptools, 4
 miniball, 18

 n.comp.include (*nb2graph*), 19
 n.comp.nb, 20
 nb2graph, 19
 neighbors (*nb2graph*), 19

 profilePlans, 3, 21, 32

 readBardCheckpoint, 3, 18, 23, 25
 readBardImage, 3, 18, 24, 24
 refineAnnealPlan, 3, 8, 10, 13, 15, 22
 refineAnnealPlan
 (*refineGreedyPlan*), 25
 refineGenoudPlan, 3, 8, 10, 13, 15, 22,
 32
 refineGenoudPlan
 (*refineGreedyPlan*), 25
 refineGRASPPlan, 3
 refineGRASPPlan
 (*refineGreedyPlan*), 25
 refineGreedyPlan, 3, 8, 10, 13, 15, 22,
 25
 refineNelderPlan, 3, 8, 10, 13, 15, 22
 refineNelderPlan
 (*refineGreedyPlan*), 25
 refineTabuPlan, 3, 8, 10
 refineTabuPlan
 (*refineGreedyPlan*), 25
 reportPlans, 3, 28, 30

 samplePlans, 3, 32
 samplePlans (*profilePlans*), 21
 scorePlans, 3, 29, 30
 spdep, 4
 startBardCluster, 3, 22, 31
 stopBardCluster
 (*startBardCluster*), 31

 writeBardCheckpoint, 3, 18, 25
 writeBardCheckpoint
 (*readBardCheckpoint*), 23
 writeBardImage, 3, 18, 24
 writeBardImage (*readBardImage*), 24