Package 'epistasisGA'

April 7, 2023

Type Package

Title An R package to identify multi-snp effects in nuclear family studies using the GADGETS method

Version 1.0.2

Description This package runs the GADGETS method to identify epistatic effects in nuclear family studies. It also provides functions for permutation-based inference and graphical visualization of the results.

License GPL-3

Encoding UTF-8

Imports BiocParallel, data.table, matrixStats, stats, survival, igraph, batchtools, qgraph, grDevices, parallel, Rcpp (>= 0.11.0), ggplot2, grid, graphics, utils

Suggests Matrix, BiocStyle, knitr, rmarkdown, magrittr, kableExtra, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.2.1

Depends R (>= 4.0)

biocViews Genetics, SNP, GeneticVariability

VignetteBuilder knitr

URL https://github.com/mnodzenski/epistasisGA

BugReports https://github.com/mnodzenski/epistasisGA/issues

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/epistasisGA

git_branch RELEASE_3_16

git_last_commit 89c2598

git_last_commit_date 2023-01-19

Date/Publication 2023-04-07

Author Michael Nodzenski [aut, cre], Juno Krahn [ctb]

Maintainer Michael Nodzenski <michael.nodzenski@gmail.com>

R topics documented:

case 2
chrom.fitness.score 3
combine.islands 6
compute.graphical.scores 7
dad 10
epistasis.test 11
epistasisGA 13
GADGETS 13
global.test 17
mom 22
network.plot 23
permute.dataset 26
preprocess.genetic.data 27
run.gadgets 30
snp.annotations 33

Index **34**

`case` *SNP counts for the affected children of case-parent triads.*

Description

A simulated dataset containing the counts of the alternate allele for 100 SNPs for the affected child in 1000 simulated case-parent triads. Columns represent SNPs, rows are individuals. SNPs in columns 51, 52, 76, and 77 represent a true risk pathway.

Usage

```
data(case)
```

Format

A data frame with 1000 rows and 100 variables

`chrom.fitness.score` *A function to assign a fitness score to a chromosome*

Description

This function assigns a fitness score to a chromosome. It is a wrapper for the Rcpp function chrom_fitness_score.

Usage

```
chrom.fitness.score(
  case.genetic.data,
  complement.genetic.data,
  case.comp.differences,
  target.snps,
  cases.minus.complements,
  both.one.mat,
  block.ld.mat,
  weight.lookup,
  case2.mat,
  case0.mat,
  comp2.mat,
  comp0.mat,
  n.different.snps.weight = 2,
  n.both.one.weight = 1,
  recessive.ref.prop = 0.75,
  recode.test.stat = 1.64,
  epi.test = FALSE,
  dif.coding = FALSE
)
```

Arguments

`case.genetic.data`
: The genetic data of the disease affected children from case-parent trios or affected/unaffected sibling pairs. Columns are SNP allele counts, and rows are individuals. The ordering of the columns must be consistent with the LD structure specified in `block.ld.mat`.

`complement.genetic.data`
: A genetic dataset from the complements of the cases, where `complement.genetic.data` = mother SNP counts + father SNP counts - case SNP counts. Columns are SNP allele counts, rows are families. If using affected/unaffected sibling pairs, this should contain the unaffected sibling genotypes.

`case.comp.differences`
: A data frame or matrix indicating `case.genetic.data` != `complement.genetic.data`, where rows correspond to individuals and columns correspond to snps.

`target.snps`	A numeric vector of the columns corresponding to the collection of SNPs, or chromosome, for which the fitness score will be computed.
`cases.minus.complements`	A matrix equal to `case.genetic.data` - `complement genetic data`.
`both.one.mat`	A matrix whose elements indicate whether both the case and complement have one copy of the minor allele, equal to `case.genetic.data == 1 & complement.genetic.data == 1`.
`block.ld.mat`	A logical, block diagonal matrix indicating whether the SNPs in `case.genetic.data` should be considered to be in linkage disequilibrium. Note that this means the ordering of the columns (SNPs) in `case.genetic.data` must be consistent with the LD blocks specified in `ld.block.mat`. In the absence of outside information, a reasonable default is to consider SNPs to be in LD if they are located on the same biological chromosome.
`weight.lookup`	A vector that maps a family weight to the weighted sum of the number of different SNPs and SNPs both equal to one.
`case2.mat`	A logical matrix indicating whether, for each SNP, the case carries 2 copies of the minor allele.
`case0.mat`	A logical matrix indicating whether, for each SNP, the case carries 0 copies of the minor allele.
`comp2.mat`	A logical matrix indicating whether, for each SNP, the complement/unaffected sibling carries 2 copies of the minor allele.
`comp0.mat`	A logical matrix indicating whether, for each SNP, the complement/unaffected sibling carries 0 copies of the minor allele.
`n.different.snps.weight`	The number by which the number of different SNPs between a case and complement/unaffected sibling is multiplied in computing the family weights. Defaults to 2.
`n.both.one.weight`	The number by which the number of SNPs equal to 1 in both the case and complement/unaffected sibling is multiplied in computing the family weights. Defaults to 1.
`recessive.ref.prop`	The proportion to which the observed proportion of informative cases with the provisional risk genotype(s) will be compared to determine whether to recode the SNP as recessive. Defaults to 0.75.
`recode.test.stat`	For a given SNP, the minimum test statistic required to recode and recompute the fitness score using recessive coding. Defaults to 1.64. See the GADGETS paper for specific details.
`epi.test`	A logical indicating whether the function should return the information required to run function `epistasis.test`. for a given SNP. See the GADGETS paper for specific details on the implementation of this argument.
`dif.coding`	A logical indicating whether, for a given SNP, the case - complement genotype difference should be coded as the sign of the difference (defaulting to false) or the raw difference Defaults to FALSE.

Value

A list:

fitness_score The chromosome fitness score.

sum_dif_vecs The weighted mean difference vector corresponding to the chromosome, with each element divided by it's pseudo-standard error. The magnitudes of these values are not particularly important, but the sign is useful. A positive value for a given SNP indicates the minor allele is positively associated with disease status, while a negative value implies the reference ('wild type') allele is positively associated with the disease.

q The number of cases with a risk-related genotype at each locus over the total number of cases or controls that have a full set of risk genotypes at each locus, among families where only one of the case or control has the full risk set.

risk_set_alleles A vector indicating the number risk alleles a case or complement must have for each SNP in `target.snps` for the case or complement to be classified as having the proposed risk set. '1+' indicates at least one copy of the risk allele is required, while '2' indicates 2 copies are needed. The risk allele can be determined based on the signs of the elements of `sum_dif_vecs`, where a negative value indicates the major allele for a given SNP is the risk allele, while a positive value implicates the minor allele.

inf_families An integer vector of the informative family rows. Only returned if `epi.test` = TRUE.

Examples

```
data(case)
data(dad)
data(mom)
case <- as.matrix(case)
dad <- as.matrix(dad)
mom <- as.matrix(mom)
comp <- mom + dad - case
case.comp.diff <- case != comp
case.minus.comp <- case - comp
storage.mode(case.minus.comp) <- "integer"
both.one.mat <- case == 1 & comp == 1
case2.mat <- case == 2
case0.mat <- case == 0
comp2.mat <- comp == 2
comp0.mat <- comp == 0
library(Matrix)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))
weight.lookup <- vapply(seq_len(6), function(x) 2^x, 1)
storage.mode(weight.lookup) <- "integer"
chrom.fitness.score(
    case, comp, case.comp.diff, c(1, 4, 7),
    case.minus.comp, both.one.mat,
    block.ld.mat, weight.lookup,
    case2.mat, case0.mat, comp2.mat,
    comp0.mat
)
```

combine.islands — *A function to combine GADGETS results for individual islands into a single dataset.*

Description

This function combines GADGETS results for individual islands into a single dataset.

Usage

```
combine.islands(
  results.dir,
  annotation.data,
  preprocessed.list,
  n.top.chroms.per.island = 1
)
```

Arguments

`results.dir`	The directory in which individual island results from `run.gadgets` are saved.
`annotation.data`	A data frame containing columns 'RSID', 'REF' and 'ALT'. Column 'RSID' gives the RSIDs for the input SNPs, with the rows ordered such that the first RSID entry corresponds to the first SNP column in the data passed to function `preprocess.genetic.data`, the second RSID corresponds to the second SNP column, etc.
`preprocessed.list`	The initial list produced by function `preprocess.genetic.data`.
`n.top.chroms.per.island`	The number of top chromosomes per island to save in the final combined list. Defaults to the top scorer.

Value

A data.table containing the results aggregated across islands. Note these results be written to `results.dir` as 'combined.island.unique.chromosome.results.rds'. See the package vignette for more detailed descriptions of the content of each output column.

Examples

```
data(case)
data(dad)
data(mom)
data(snp.annotations)
library(Matrix)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))

pp.list <- preprocess.genetic.data(case[, 1:10],
    father.genetic.data = dad[, 1:10],
    mother.genetic.data = mom[, 1:10],
    block.ld.mat = block.ld.mat[, 1:10]
)

run.gadgets(pp.list,
    n.chromosomes = 4, chromosome.size = 3, results.dir = "tmp",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)

combined.res <- combine.islands("tmp", snp.annotations[1:10, ], pp.list, 1)

unlink("tmp", recursive = TRUE)
unlink("tmp_reg", recursive = TRUE)
```

`compute.graphical.scores` *A function to compute SNP-pair scores for network plots of results.*

Description

This function returns a data.table of graphical SNP-pair scores for use in network plots of GADGETS results.

Usage

```
compute.graphical.scores(
  results.list,
  pp.list,
  score.type = "logsum",
  pval.thresh = 0.05,
  n.permutes = 10000,
  n.different.snps.weight = 2,
  n.both.one.weight = 1,
  weight.function.int = 2,
  recessive.ref.prop = 0.75,
  recode.test.stat = 1.64,
  dif.coding = FALSE,
  bp.param = bpparam()
)
```

Arguments

`results.list`	A list of length d, where d is the number of chromosome sizes to be included in the network plot. Each element of the list must be a data.table from `combine.islands` for a given chromosome size. Each data.table in the list should be subset to the top `n.top.scores` scores, otherwise an error will be returned.
`pp.list`	The list output by `preprocess.genetic.data` run on the observed data.
`score.type`	A character string specifying the method for aggregating SNP-pair scores across chromosome sizes. Options are 'max', 'sum', or 'logsum', defaulting to "logsum". For a given SNP-pair, it's graphical score will be the `score.type` of all graphical scores of chromosomes containing that pair across chromosome sizes. The choice of 'logsum' rather than 'sum' may be useful in cases where there are multiple risk-sets, and one is found much more frequently. However, it may be of interest to examine plots using both `score.type` approaches. Note that "logsum" is actually the log of one plus the sum of the SNP-pair scores to avoid nodes or edges having negative weights.
`pval.thresh`	A numeric value between 0 and 1 specifying the epistasis test p-value threshold for a chromosome to contribute to the network. Any chromosomes with epistasis p-value greater than `pval.thresh` will not contribute to network plots. The argument defaults to 0.05. It must be <= 0.6 (to ensure positive scores).
`n.permutes`	The number of permutations on which to base the epistasis tests. Defaults to 10000.
`n.different.snps.weight`	The number by which the number of different SNPs between a case and complement/unaffected sibling is multiplied in computing the family weights. Defaults to 2.
`n.both.one.weight`	The number by which the number of SNPs equal to 1 in both the case and complement/unaffected sibling is multiplied in computing the family weights. Defaults to 1.
`weight.function.int`	An integer used to assign family weights. Specifically, we use `weight.function.int` in a function that takes the weighted sum of the number of different SNPs and SNPs both equal to one as an argument, denoted as x, and returns a family weight equal to `weight.function.int^x`. Defaults to 2.

`recessive.ref.prop`
: The proportion to which the observed proportion of informative cases with the provisional risk genotype(s) will be compared to determine whether to recode the SNP as recessive. Defaults to 0.75.

`recode.test.stat`
: For a given SNP, the minimum test statistic required to recode and recompute the fitness score using recessive coding. Defaults to 1.64. See the GADGETS paper for specific details.

`dif.coding`
: A logical indicating whether, for a given SNP, the case - complement genotype difference should be coded as the sign of the difference (defaulting to false) or the raw difference.

`bp.param`
: The BPPARAM argument to be passed to bplapply. See `BiocParallel::bplapply` for more details.

Value

A list of two elements:

pair.scores A data.table containing SNP-pair graphical scores, where the first four columns represent SNPs and the fifth column (pair.score) is the graphical SNP-pair score.

snp.scores A data.table containing individual SNP graphical scores, where the first two columns represent SNPs and the third column (snp.score) is the graphical SNP score.

Examples

```
data(case)
data(dad)
data(mom)
data(snp.annotations)
library(Matrix)
set.seed(1400)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))

# preprocess data
target.snps <- c(1:3, 30:32, 60:62, 85)
pp.list <- preprocess.genetic.data(case[, target.snps],
    father.genetic.data = dad[, target.snps],
    mother.genetic.data = mom[, target.snps],
    block.ld.mat = block.ld.mat[target.snps, target.snps]
)
## run GA for observed data

# observed data chromosome size 2
run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 2, results.dir = "tmp_2",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
combined.res2 <- combine.islands(
    "tmp_2", snp.annotations[target.snps, ],
    pp.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# observed data chromosome size 3
run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 3, results.dir = "tmp_3",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
combined.res3 <- combine.islands(
    "tmp_3", snp.annotations[target.snps, ],
    pp.list, 2
)
unlink("tmp_reg", recursive = TRUE)

## create list of results

final.results <- list(combined.res2[1:3, ], combined.res3[1:3, ])

## compute edge scores
edge.dt <- compute.graphical.scores(final.results, pp.list,
    pval.thresh = 0.5
)

lapply(c("tmp_2", "tmp_3"), unlink, recursive = TRUE)
```

---

dad *SNP counts for the fathers of case-parent triads.*

Description

A simulated dataset containing the counts of the alternate allele for 100 SNPs for the fathers in 1000 simulated case-parent triads. Columns represent SNPs, rows are individuals. SNPs in columns 51, 52, 76, and 77 represent a true risk pathway.

Usage

```
data(dad)
```

Format

A data frame with 1000 rows and 100 variables

`epistasis.test` *A function to run a test of the null hypothesis that a collection of SNPs do not exhibit epistasis, conditional upon observed marginal SNP-disease associations.*

Description

This function runs a permutation based test of the null hypothesis that a collection of SNPs do not exhibit epistasis, conditional upon observed marginal SNP-disease associations.

Usage

```
epistasis.test(
  snp.cols,
  preprocessed.list,
  n.permutes = 10000,
  n.different.snps.weight = 2,
  n.both.one.weight = 1,
  weight.function.int = 2,
  recessive.ref.prop = 0.75,
  recode.test.stat = 1.64,
  dif.coding = FALSE
)
```

Arguments

`snp.cols`	An integer vector specifying the columns in the input data containing the SNPs to be tested.
`preprocessed.list`	The initial list produced by function `preprocess.genetic.data`.
`n.permutes`	The number of permutations on which to base the test. Defaults to 1000.
`n.different.snps.weight`	The number by which the number of different SNPs between a case and complement/unaffected sibling is multiplied in computing the family weights. Defaults to 2.
`n.both.one.weight`	The number by which the number of SNPs equal to 1 in both the case and complement/unaffected sibling is multiplied in computing the family weights. Defaults to 1.

`weight.function.int`
: An integer used to assign family weights. Specifically, we use `weight.function.int` in a function that takes the weighted sum of the number of different SNPs and SNPs both equal to one as an argument, denoted as `x`, and returns a family weight equal to `weight.function.int^x`. Defaults to 2.

`recessive.ref.prop`
: The proportion to which the observed proportion of informative cases with the provisional risk genotype(s) will be compared to determine whether to recode the SNP as recessive. Defaults to 0.75.

`recode.test.stat`
: For a given SNP, the minimum test statistic required to recode and recompute the fitness score using recessive coding. Defaults to 1.64. See the GADGETS paper for specific details.

`dif.coding`
: A logical indicating whether, for a given SNP, the case - complement genotype difference should be coded as the sign of the difference (defaulting to false) or the raw difference.

Value

A list of thee elements:

pval The p-value of the test.

obs.fitness.score The fitness score from the observed data

perm.fitness.scores A vector of fitness scores for the permuted datasets.

Examples

```
data(case)
data(dad)
data(mom)
data(snp.annotations)
library(Matrix)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))

pp.list <- preprocess.genetic.data(case,
    father.genetic.data = dad,
    mother.genetic.data = mom,
    block.ld.mat = block.ld.mat
)

run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 3,
    results.dir = "tmp", cluster.type = "interactive",
    registryargs = list(file.dir = "tmp_reg", seed = 1300),
    n.islands = 8, island.cluster.size = 4,
    n.migrations = 2
)

combined.res <- combine.islands("tmp", snp.annotations, pp.list, 2)

top.snps <- as.vector(t(combined.res[1, 1:3]))
set.seed(10)
epi.test.res <- epistasis.test(top.snps, pp.list)

unlink("tmp", recursive = TRUE)
unlink("tmp_reg", recursive = TRUE)
```

`epistasisGA` `epistasisGA` *package*

Description

A package implementing the GADGETS method to detect multi-SNP effects in case-parent triad or affected/unaffected sibling studies.

`GADGETS` *A function to run the GADGETS method*

Description

This function runs the GADGETS method on a given cluster of islands. It is a wrapper for the underlying Rcpp function run_GADGETS.

Usage

```
GADGETS(
  cluster.number,
  results.dir,
  case.genetic.data,
  complement.genetic.data,
  case.comp.different,
  case.minus.comp,
  both.one.mat,
  block.ld.mat,
  n.chromosomes,
  chromosome.size,
  snp.chisq,
  original.col.numbers,
  weight.lookup,
  case2.mat,
  case0.mat,
  comp2.mat,
  comp0.mat,
  island.cluster.size = 4,
  n.migrations = 20,
  n.different.snps.weight = 2,
  n.both.one.weight = 1,
  migration.interval = 50,
  gen.same.fitness = 50,
  max.generations = 500,
  initial.sample.duplicates = FALSE,
  crossover.prop = 0.8,
  recessive.ref.prop = 0.75,
  recode.test.stat = 1.64,
  dif.coding = FALSE
)
```

Arguments

`cluster.number`	An integer indicating the cluster number (used for labeling the output file).
`results.dir`	The directory to which island results will be saved.
`case.genetic.data`	The genetic data of the disease affected children from case-parent trios or affected/unaffected sibling pairs. Columns are SNP allele counts, and rows are individuals. The ordering of the columns must be consistent with the LD structure specified in `block.ld.mat`.
`complement.genetic.data`	A genetic dataset from the complements of the cases, where `complement.genetic.data` = mother SNP counts + father SNP counts - case SNP counts. Columns are SNP allele counts, rows are families. If using affected/unaffected sibling pairs, this should contain the unaffected sibling genotypes.
`case.comp.different`	A data frame or matrix indicating `case.genetic.data` != `complement.genetic.data`, where rows correspond to individuals and columns correspond to snps.
`case.minus.comp`	A matrix equal to `case.genetic.data` - `complement genetic data`.
`both.one.mat`	A matrix whose elements indicate whether both the case and complement have one copy of the minor allele, equal to `case.genetic.data == 1 & complement.genetic.data == 1`.
`block.ld.mat`	A logical, block diagonal matrix indicating whether the SNPs in `case.genetic.data` should be considered to be in linkage disequilibrium. Note that this means the ordering of the columns (SNPs) in `case.genetic.data` must be consistent with the LD blocks specified in `ld.block.mat`. In the absence of outside information, a reasonable default is to consider SNPs to be in LD if they are located on the same biological chromosome.
`n.chromosomes`	An integer specifying the number of chromosomes to use in the GA.

`chromosome.size`	An integer specifying the number of SNPs on each chromosome.
`snp.chisq`	A vector of statistics to be used in sampling SNPs for mutation. By default, these are the square roots of the chi-square marginal SNP-disease association statistics for each column in `case.genetic.data`, but can also be manually specified or uniformly 1 (corresponding to totally random sampling).
`original.col.numbers`	A vector of integers indicating the original column number of each SNP in `case.genetic.data`. This is needed due to removal of low frequency SNPs in `preprocess.genetic.data`.
`weight.lookup`	A vector that maps a family weight to the weighted sum of the number of different SNPs and SNPs both equal to one.
`case2.mat`	A logical matrix indicating whether, for each SNP, the case carries 2 copies of the minor allele.
`case0.mat`	A logical matrix indicating whether, for each SNP, the case carries 0 copies of the minor allele.
`comp2.mat`	A logical matrix indicating whether, for each SNP, the complement/unaffected sibling carries 2 copies of the minor allele.
`comp0.mat`	A logical matrix indicating whether, for each SNP, the complement/unaffected sibling carries 0 copies of the minor allele.
`island.cluster.size`	An integer specifying the number of islands in the cluster. See coderun.gadgets for additional details.
`n.migrations`	The number of chromosomes that migrate among islands. This value must be less than `n.chromosomes` and greater than 0, defaulting to 20.
`n.different.snps.weight`	The number by which the number of different SNPs between a case and complement is multiplied in computing the family weights. Defaults to 2.
`n.both.one.weight`	The number by which the number of SNPs equal to 1 in both the case and complement is multiplied in computing the family weights. Defaults to 1.
`migration.interval`	The interval of generations for which GADGETS will run prior to migration of top chromosomes among islands in a cluster. Defaults to 50. In other words, top chromosomes will migrate among cluster islands every `migration.interval` generations. We also check for convergence at each of these intervals.
`gen.same.fitness`	The number of consecutive generations with the same fitness score required for algorithm termination. Defaults to 50.
`max.generations`	The maximum number of generations for which GADGETS will run. Defaults to 500.
`initial.sample.duplicates`	A logical indicating whether the same SNP can appear in more than one chromosome in the initial sample of chromosomes (the same SNP may appear in more than one chromosome thereafter, regardless) . Defaults to FALSE.

`crossover.prop`	A numeric between 0 and 1 indicating the proportion of chromosomes to be subjected to cross over. The remaining proportion will be mutated. Defaults to 0.8.
`recessive.ref.prop`	The proportion to which the observed proportion of informative cases with the provisional risk genotype(s) will be compared to determine whether to recode the SNP as recessive. Defaults to 0.75.
`recode.test.stat`	For a given SNP, the minimum test statistic required to recode and recompute the fitness score using recessive coding. Defaults to 1.64. See the GADGETS paper for specific details.
`dif.coding`	A logical indicating whether, for a given SNP, the case - complement genotype difference should be coded as the sign of the difference (defaulting to false) or the raw difference.

Value

For each island in the cluster, an rds object containing a list with the following elements will be written to `results.dir`:

top.chromosome.results A data.table of the final generation chromosomes, their fitness scores, their difference vectors, and the number of risk alleles required for each chromosome SNP for a case or complement to be classified as having the provisional risk set. See the package vignette for an example and the documentation for `chrom.fitness.score` for additional details.

n.generations The total number of generations run.

Examples

```
set.seed(10)
data(case)
data(dad)
data(mom)
library(Matrix)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))
data.list <- preprocess.genetic.data(case[, 1:10],
    father.genetic.data = dad[, 1:10],
    mother.genetic.data = mom[, 1:10],
    block.ld.mat = block.ld.mat[1:10, 1:10]
)

case.genetic.data <- as.matrix(data.list$case.genetic.data)
complement.genetic.data <- as.matrix(data.list$complement.genetic.data)
original.col.numbers <- data.list$original.col.numbers
chisq.stats <- data.list$chisq.stats
```

```
block.ld.mat <- data.list$block.ld.mat
case.minus.comp <- as.matrix(case.genetic.data -
    complement.genetic.data)
case.comp.different <- case.minus.comp != 0
both.one.mat <- complement.genetic.data == 1 & case.genetic.data == 1
case2.mat <- case.genetic.data == 2
case0.mat <- case.genetic.data == 0
comp2.mat <- complement.genetic.data == 2
comp0.mat <- complement.genetic.data == 0
snp.chisq <- sqrt(chisq.stats)
weight.lookup <- vapply(seq_len(6), function(x) 2^x, 1)
dir.create("tmp")
GADGETS(
    cluster.number = 1, results.dir = "tmp",
    case.genetic.data = case.genetic.data,
    complement.genetic.data = complement.genetic.data,
    case.comp.different = case.comp.different,
    case.minus.comp = case.minus.comp, both.one.mat = both.one.mat,
    block.ld.mat = block.ld.mat, n.chromosomes = 10,
    chromosome.size = 3, snp.chisq = snp.chisq,
    original.col.numbers = original.col.numbers,
    weight.lookup = weight.lookup, case2.mat = case2.mat,
    case0.mat = case0.mat, comp2.mat = comp2.mat,
    comp0.mat = comp0.mat, n.migrations = 2,
    migration.interval = 5, max.generations = 10
)
unlink("tmp", recursive = TRUE)
```

global.test — *A function to run a global test of the null hypothesis that there are no SNP-disease associations across a range of chromosome sizes*

Description

This function runs a global test of the null hypothesis that there are no SNP-disease associations across a range of chromosome sizes

Usage

```
global.test(results.list, n.top.scores = 10)
```

Arguments

`results.list`	A list of length d, where d is the number of chromosome sizes to be included in a global test. Each element of the list must itself be a list whose first element `observed.data` is a vector of fitness scores from `combine.islands` for a given chromosome size. The second element `permutation.list` is a list containing vectors of all permutation results fitness scores, again using the results output by `combine.islands` for each permutation.

`n.top.scores` The number of top scoring chromosomes, for each chromosome size, to be used in calculating the global test. Defaults to 10.

Value

A list containing the following:

obs.test.stat The observed test statistic.

perm.test.stats A vector of test statistics from permuted data.

pval The p-value for the global test.

obs.marginal.test.stats A vector of observed test statistics for each chromosome size.

perm.marginal.test.stats.mat A matrix of test statistics for the permutation datasets, where rows correspond to permutations and columns correspond to chromosome sizes.

marginal.pvals A vector containing marignal p-values for each chromosome size.

max.obs.fitness A vector of the maximum fitness score for each chromosome size in the observed data.

max.perm.fitness A list of vectors for each chromosome size of maximum observed fitness scores for each permutation.

max.order.pvals A vector of p-values for the maximum observed order statistics for each chromosome size. P-values are one plus the number of permutation based maximum order statistics that exceed the observed maximum fitness score divided by the total number of permutations plus one.

boxplot.grob A grob of a ggplot plot of the observed vs permuted fitness score densities for each chromosome size.

chrom.size.k A vector indicating the number of top scores (k) from each chromosome size that the test used. This will be equal to `n.top.scores` unless GADGETS returns fewer than `n.top.scores` unique chromosomes for the observed data or any permute, in which case the chromosome size-specific value will be equal to the smallest number of unique chromosomes returned.

max.perm.95th.pctl The 95th percentile of the permutation maximum order statistics for each chromosome size.

Examples

```
data(case)
data(dad)
data(mom)
data(snp.annotations)
library(Matrix)
set.seed(1400)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))
```

```
pp.list <- preprocess.genetic.data(case[, 1:10],
    father.genetic.data = dad[, 1:10],
    mother.genetic.data = mom[, 1:10],
    block.ld.mat = block.ld.mat[1:10, 1:10]
)
## run GA for observed data

# observed data chromosome size 2
run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 2, results.dir = "tmp_2",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
combined.res2 <- combine.islands(
    "tmp_2", snp.annotations[1:10, ], pp.list,
    2
)
unlink("tmp_reg", recursive = TRUE)

# observed data chromosome size 3
run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 3, results.dir = "tmp_3",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
combined.res3 <- combine.islands(
    "tmp_3", snp.annotations[1:10, ], pp.list,
    2
)
unlink("tmp_reg", recursive = TRUE)

# create three permuted datasets
set.seed(1400)
perm.data.list <- permute.dataset(case[, 1:10],
    father.genetic.data = dad[, 1:10],
    mother.genetic.data = mom[, 1:10],
    n.permutations = 3
)

# pre-process permuted data
p1.list <- preprocess.genetic.data(perm.data.list[["permutation1"]]$case,
    complement.genetic.data = perm.data.list[["permutation1"]]$comp,
    block.ld.mat = block.ld.mat[1:10, 1:10]
)
```

```
p2.list <- preprocess.genetic.data(perm.data.list[["permutation2"]]$case,
    complement.genetic.data = perm.data.list[["permutation2"]]$comp,
    block.ld.mat = block.ld.mat[1:10, 1:10]
)

p3.list <- preprocess.genetic.data(perm.data.list[["permutation3"]]$case,
    complement.genetic.data = perm.data.list[["permutation3"]]$comp,
    block.ld.mat = block.ld.mat[1:10, 1:10]
)

## run GA for permuted data

# permutation 1, chromosome size 2
run.gadgets(p1.list,
    n.chromosomes = 5, chromosome.size = 2, results.dir = "p1_tmp_2",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
p1.combined.res2 <- combine.islands(
    "p1_tmp_2", snp.annotations[1:10, ],
    p1.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# permutation 1, chromosome size 3
run.gadgets(p1.list,
    n.chromosomes = 5, chromosome.size = 3, results.dir = "p1_tmp_3",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
p1.combined.res3 <- combine.islands(
    "p1_tmp_3", snp.annotations[1:10, ],
    p1.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# permutation 2, chromosome size 2
run.gadgets(p2.list,
    n.chromosomes = 5, chromosome.size = 2, results.dir = "p2_tmp_2",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
p2.combined.res2 <- combine.islands(
    "p2_tmp_2", snp.annotations[1:10, ],
    p2.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# permutation 2, chromosome size 3
run.gadgets(p2.list,
    n.chromosomes = 5, chromosome.size = 3, results.dir = "p2_tmp_3",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
p2.combined.res3 <- combine.islands(
    "p2_tmp_3", snp.annotations[1:10, ],
    p2.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# permutation 3, chromosome size 2
run.gadgets(p3.list,
    n.chromosomes = 5, chromosome.size = 2, results.dir = "p3_tmp_2",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
p3.combined.res2 <- combine.islands(
    "p3_tmp_2", snp.annotations[1:10, ],
    p3.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# permutation 3, chromosome size 3
run.gadgets(p3.list,
    n.chromosomes = 5, chromosome.size = 3, results.dir = "p3_tmp_3",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
p3.combined.res3 <- combine.islands(
    "p3_tmp_3", snp.annotations[1:10, ],
    p3.list, 2
)
```

```
unlink("tmp_reg", recursive = TRUE)

## create list of results

# chromosome size 2 results
chrom2.list <- list(
    observed.data = combined.res2$fitness.score,
    permutation.list = list(
        p1.combined.res2$fitness.score,
        p2.combined.res2$fitness.score,
        p3.combined.res2$fitness.score
    )
)

# chromosome size 3 results
chrom3.list <- list(
    observed.data = combined.res3$fitness.score,
    permutation.list = list(
        p1.combined.res3$fitness.score,
        p2.combined.res3$fitness.score,
        p3.combined.res3$fitness.score
    )
)

final.results <- list(chrom2.list, chrom3.list)

## run global test
global.test.res <- global.test(final.results, 1)

lapply(c(
    "tmp_2", "tmp_3", "p1_tmp_2", "p2_tmp_2", "p3_tmp_2",
    "p1_tmp_3", "p2_tmp_3", "p3_tmp_3"
), unlink, recursive = TRUE)
```

mom *SNP counts for the mothers of case-parent triads.*

Description

A simulated dataset containing the counts of the alternate allele for 100 SNPs for the mothers in 1000 simulated case-parent triads. Columns represent SNPs, rows are individuals. SNPs in columns 51, 52, 76, and 77 represent a true risk pathway.

Usage

```
data(mom)
```

Format

A data frame with 1000 rows and 100 variables

`network.plot` *A function to plot a network of SNPs with potential multi-SNP effects.*

Description

This function plots a network of SNPs with potential multi-SNP effects.

Usage

```
network.plot(
  graphical.score.list,
  preprocessed.list,
  score.type = "logsum",
  n.top.scoring.pairs = NULL,
  node.shape = "circle",
  repulse.rad = 1000,
  node.size = 25,
  graph.area = 100,
  vertex.label.cex = 0.5,
  edge.width.cex = 12,
  plot = TRUE,
  edge.color.ramp = c("lightblue", "blue"),
  node.color.ramp = c("white", "red"),
  plot.legend = TRUE,
  high.ld.threshold = 0.1,
  plot.margins = c(2, 1, 2, 1),
  legend.title.cex = 1.75,
  legend.axis.cex = 1.75,
  ...
)
```

Arguments

`graphical.score.list`
: The list returned by function `compute.graphical.scores`, or a subset of it. By default, the SNPs will be labeled with their RSIDs, listed in columns 3 and 4. Users can create custom labels by changing the values in these two columns.

`preprocessed.list`
: The initial list produced by function `preprocess.genetic.data`.

`score.type`
: A character string specifying the method for aggregating SNP-pair scores across chromosome sizes. Options are 'max', 'sum', or 'logsum', defaulting to "logsum". For a given SNP-pair, it's graphical score will be the `score.type` of all graphical scores of chromosomes containing that pair across chromosome sizes. Pair scores will be proportional to the sum of graphical scores for either 'logsum' or 'sum', but 'logsum' may be useful in cases where there are multiple

risk-sets, and one is found much more frequently. Note that "logsum" is actually the log of one plus the sum of the SNP-pair scores to avoid nodes or edges having negative weights.

`n.top.scoring.pairs`	An integer indicating the number of top scoring pairs to plot. Defaults to, NULL, which plots all pairs. For large networks, plotting a subset of the top scoring pairs can improve the appearance of the graph.
`node.shape`	The desired node shape. See `names(igraph:::.igraph.shapes)` for available shapes. Defaults to circle.
`repulse.rad`	A scalar affecting the graph shape. Decrease to reduce overlapping nodes, increase to move nodes closer together.
`node.size`	A scalar affecting the size of the graph nodes. Increase to increase size.
`graph.area`	A scalar affecting the size of the graph area. Increase to increase graph area.
`vertex.label.cex`	A scalar controlling the size of the vertex label. Increase to increase size.
`edge.width.cex`	A scalar controlling the width of the graph edges. Increase to make edges wider.
`plot`	A logical indicating whether the network should be plotted. If set to false, this function will return an igraph object to be used for manual plotting.
`edge.color.ramp`	A character vector of colors. The coloring of the network edges will be shown on a gradient, with the lower scoring edge weights closer to the first color specified in `edge.color.ramp`, and higher scoring weights closer to the last color specified. By default, the low scoring edges are light blue, and high scoring edges are dark blue.
`node.color.ramp`	A character vector of colors. The coloring of the network nodes will be shown on a gradient, with the lower scoring nodes closer to the first color specified in `node.color.ramp`, and higher scoring nodes closer to the last color specified. By default, the low scoring nodes are whiter, and high scoring edges are redder.
`plot.legend`	A boolean indicating whether a legend should be plotted. Defaults to TRUE.
`high.ld.threshold`	A numeric value between 0 and 1, indicating the r^2 threshold in complements (or unaffected siblings) above which a pair of SNPs in the same LD block (as specified in `preprocessed.list`) should be considered in high LD. Connections between these high LD SNPs will be dashed instead of solid lines. Defaults to 0.1.
`plot.margins`	A vector of length 4 passed to `par(mar = )`. Defaults to c(2, 1, 2, 1).
`legend.title.cex`	A numeric value controlling the size of the legend titles. Defaults to 1.75. Increase to increase font size, decrease to decrease font size.
`legend.axis.cex`	A numeric value controlling the size of the legend axis labels. Defaults to 1.75. Increase to increase font size, decrease to decrease font size.
`...`	Additional arguments to be passed to `plot.igraph`.

Value

An igraph object, if `plot` is set to FALSE.

Examples

```
data(case)
data(dad)
data(mom)
data(snp.annotations)
library(Matrix)
set.seed(1400)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))

# preprocess data
target.snps <- c(1:3, 30:32, 60:62, 85)
pp.list <- preprocess.genetic.data(case[, target.snps],
    father.genetic.data = dad[, target.snps],
    mother.genetic.data = mom[, target.snps],
    block.ld.mat = block.ld.mat[target.snps, target.snps]
)
## run GA for observed data

# observed data chromosome size 2
run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 2, results.dir = "tmp_2",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
combined.res2 <- combine.islands(
    "tmp_2", snp.annotations[target.snps, ],
    pp.list, 2
)
unlink("tmp_reg", recursive = TRUE)

# observed data chromosome size 3
run.gadgets(pp.list,
    n.chromosomes = 5, chromosome.size = 3, results.dir = "tmp_3",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)
combined.res3 <- combine.islands(
    "tmp_3", snp.annotations[target.snps, ],
    pp.list, 2
)
unlink("tmp_reg", recursive = TRUE)

## create list of results
final.results <- list(combined.res2[1:3, ], combined.res3[1:3, ])

## compute edge scores
set.seed(20)
graphical.list <- compute.graphical.scores(final.results, pp.list,
    pval.thresh = 0.5
)

## plot
set.seed(10)
network.plot(graphical.list, pp.list)

lapply(c("tmp_2", "tmp_3"), unlink, recursive = TRUE)
```

`permute.dataset` *A function to create permuted datasets for permutation based hypothesis testing.*

Description

This function creates permuted datasets for permutation based hypothesis testing of GADGETS fitness scores.

Usage

```
permute.dataset(
  case.genetic.data,
  complement.genetic.data = NULL,
  father.genetic.data = NULL,
  mother.genetic.data = NULL,
  n.permutations = 100
)
```

Arguments

`case.genetic.data`
: The genetic data of the disease affected children. Columns are SNP allele counts, and rows are individuals.

`complement.genetic.data`	A genetic dataset from the complements of the cases, where `complement.genetic.data` = mother SNP counts + father SNP counts - case SNP counts. If using affected/unaffected sibling pairs, this should be the genetic data for the unaffected sibling. Columns are SNP allele counts, rows are families. If not specified, `father.genetic.data` and `mother.genetic.data` must be specified.
`father.genetic.data`	The genetic data for the fathers of the cases. Columns are SNP allele counts, rows are individuals. Does not need to be specified if `complement.genetic.data` is specified.
`mother.genetic.data`	The genetic data for the mothers of the cases. Columns are SNP allele counts, rows are individuals. Does not need to be specified if `complement.genetic.data` is specified.
`n.permutations`	The number of permuted datasets to create.

Value

A list of `n.permutations` pairs of case and complement data, where the observed case/complement status has been randomly flipped or not flipped.

Examples

```
data(case)
data(dad)
data(mom)
set.seed(15)
perm.data.list <- permute.dataset(case,
    father.genetic.data = dad, mother.genetic.data = mom,
    n.permutations = 2
)
```

`preprocess.genetic.data` *A function to pre-process case-parent triad of affected/unaffected sibling data.*

Description

This function performs several pre-processing steps, intended for use before function run.gadgets.

Usage

```
preprocess.genetic.data(
  case.genetic.data,
  complement.genetic.data = NULL,
  father.genetic.data = NULL,
  mother.genetic.data = NULL,
  block.ld.mat,
  min.allele.freq = 0,
  bp.param = bpparam(),
  snp.sampling.probs = NULL
)
```

Arguments

`case.genetic.data`
: The genetic data of the disease affected children from case-parent trios or affected/unaffected sibling pairs. Columns are SNP allele counts, and rows are individuals. The ordering of the columns must be consistent with the LD structure specified in `block.ld.mat`.

`complement.genetic.data`
: A genetic dataset from the complements of the cases, where `complement.genetic.data` = mother SNP counts + father SNP counts - case SNP counts. If using affected/unaffected siblings this should be the genotypes for the unaffected siblings. Columns are SNP allele counts, rows are families. If not specified, `father.genetic.data` and `mother.genetic.data` must be specified.

`father.genetic.data`
: The genetic data for the fathers of the cases. Columns are SNP allele counts, rows are individuals. Does not need to be specified if `complement.genetic.data` is specified.

`mother.genetic.data`
: The genetic data for the mothers of the cases. Columns are SNP allele counts, rows are individuals. Does not need to be specified if `complement.genetic.data` is specified.

`block.ld.mat`
: A logical, block diagonal matrix indicating whether the SNPs in `case.genetic.data` should be considered to be in linkage disequilibrium. Note that this means the ordering of the columns (SNPs) in `case.genetic.data` must be consistent with the LD blocks specified in `ld.block.mat`. In the absence of outside information, a reasonable default is to consider SNPs to be in LD if they are located on the same biological chromosome.

`min.allele.freq`
: The minimum minor allele frequency required for a SNP to be considered for inclusion in the genetic algorithm. Any SNPs with MAF < `min.allele.freq` in the parents, or the combined group of affected and unaffected siblings, will be filtered out. Defaults to 0 (no filtering).

`bp.param`
: The BPPARAM argument to be passed to bplapply when estimating marginal disease associations for each SNP. If using a cluster computer, this parameter needs to be set with care. See `BiocParallel::bplapply` for more details

`snp.sampling.probs`
: A vector indicating the sampling probabilities of the SNPs in `case.genetic.data`. SNPs will be sampled in the genetic algorithm proportional to the values specified. If not specified, by default, chi-square statistics of association will be

computed for each SNP, and sampling will be proportional to the root of these statistics. If user specified, the vector values need not sum to 1, they just need to be positive real numbers. See argument `prob` from function `sample` for more details.

Value

A list containing the following:

case.genetic.data The pre-processed version of the case genetic data. Any missing genotypes for a given family will be coded as -9 for both case and complement, resulting in that family being uninformative for that SNP.

complement.genetic.data Pre-processed complement or unaffected sibling genetic data. If mother and father data are input, the complement genetic data are first created and then pre-processed. Any missing genotypes for a given family will be coded as -9 for both case and complement, resulting in that family being uninformative for that SNP.

chisq.stats A vector of chi-square statistics corresponding to marginal SNP-disease associations, if `snp.sampling.probs` is not specified, and `snp.sampling.probs` if specified.

original.col.numbers A vector indicating the original column number of each non-filtered SNP remaining in the analysis data.

block.ld.mat The pre-processed version of `block.ld.mat`.

minor.allele.vec A vector indicating whether the alternate allele was the minor allele for each column in the input data.

Examples

```
data(case)
data(dad)
data(mom)
library(Matrix)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))
res <- preprocess.genetic.data(case[, 1:10],
    father.genetic.data = dad[, 1:10],
    mother.genetic.data = mom[, 1:10],
    block.ld.mat = block.ld.mat[, 1:10]
)
```

`run.gadgets`	*A function to run the GADGETS algorithm to detect multi-SNP effects in case-parent triad studies.*

Description

This function runs the GADGETS algorithm to detect multi-SNP effects in case-parent triad studies.

Usage

```
run.gadgets(
  data.list,
  n.chromosomes,
  chromosome.size,
  results.dir,
  cluster.type,
  registryargs = list(file.dir = NA, seed = 1500),
  resources = list(),
  cluster.template = NULL,
  n.workers = min(detectCores() - 2, n.islands/island.cluster.size),
  n.chunks = NULL,
  n.different.snps.weight = 2,
  n.both.one.weight = 1,
  weight.function.int = 2,
  generations = 500,
  gen.same.fitness = 50,
  initial.sample.duplicates = FALSE,
  snp.sampling.type = "chisq",
  crossover.prop = 0.8,
  n.islands = 1000,
  island.cluster.size = 4,
  migration.generations = 50,
  n.migrations = 20,
  recessive.ref.prop = 0.75,
  recode.test.stat = 1.64,
  dif.coding = FALSE
)
```

Arguments

`data.list`	The output list from `preprocess.genetic.data`.
`n.chromosomes`	An integer specifying the number of chromosomes to use for each island in GADGETS.
`chromosome.size`	An integer specifying the number of SNPs in each chromosome.
`results.dir`	The directory to which island results will be saved.

`cluster.type`	A character string indicating the type of cluster on which to evolve solutions in parallel. Supported options are interactive, socket, multicore, sge, slurm, lsf, openlava, or torque. See the documentation for package batchtools for more information.
`registryargs`	A list of the arguments to be provided to `batchtools::makeRegistry`.
`resources`	A named list of key-value pairs to be substituted into the template file. Options available are specified in `batchtools::submitJobs`.
`cluster.template`	A character string of the path to the template file required for the cluster specified in `cluster.type`. Defaults to NULL. Required for options sge, slurm, lsf, openlava and torque of argument `cluster.type`.
`n.workers`	An integer indicating the number of workers for the cluster specified in `cluster.type`, if socket or multicore. Defaults to `parallel::detectCores - 2`.
`n.chunks`	An integer specifying the number of chunks jobs running island clusters should be split into when dispatching jobs using `batchtools`. For multicore or socket `cluster.type`, this defaults to `n.workers`, resulting in the total number of island cluster jobs (equal to `n.islands\island.cluster.size`) being split into `n.chunks` chunks. All chunks then run in parallel, with jobs within a chunk running sequentially. For other cluster types, this defaults to 1 chunk, with the recommendation that users of HPC clusters which support array jobs specify `chunks.as.arrayjobs = TRUE` in argument `resources`. For those users, the setup will submit an array of `n.islands\island.cluster.size` jobs to the cluster. For HPC clusters that do not support array jobs, the default setting should not be used. See `batchtools::submitJobs` for more information on job chunking.
`n.different.snps.weight`	The number by which the number of different SNPs between a case and complement or unaffected sibling is multiplied in computing the family weights. Defaults to 2.
`n.both.one.weight`	The number by which the number of SNPs equal to 1 in both the case and complement or unaffected sibling is multiplied in computing the family weights. Defaults to 1.
`weight.function.int`	An integer used to assign family weights. Specifically, we use `weight.function.int` in a function that takes the weighted sum of the number of different SNPs and SNPs both equal to one as an argument, denoted as x, and returns a family weight equal to `weight.function.int`^x. Defaults to 2.
`generations`	The maximum number of generations for which GADGETS will run. Defaults to 500.
`gen.same.fitness`	The number of consecutive generations with the same fitness score required for algorithm termination. Defaults to 50.
`initial.sample.duplicates`	A logical indicating whether the same SNP can appear in more than one chromosome in the initial sample of chromosomes (the same SNP may appear in more than one chromosome thereafter, regardless) . Default to FALSE.

`snp.sampling.type`	A string indicating how SNPs are to be sampled for mutations. Options are 'chisq', 'random', or 'manual'. The 'chisq' option takes into account the marginal association between a SNP and disease status, with larger marginal associations corresponding to higher sampling probabilities.The 'random' option gives each SNP the same sampling probability regardless of marginal association. The 'manual' option should be used when `snp.sampling.probs` are manually input into function `preprocess.genetic.data`. Defaults to 'chisq'.
`crossover.prop`	A numeric between 0 and 1 indicating the proportion of chromosomes to be subjected to cross over. The remaining proportion will be mutated. Defaults to 0.8.
`n.islands`	An integer indicating the number of islands to be used. Defaults to 1000.
`island.cluster.size`	An integer specifying the number of islands in a given cluster. Must evenly divide `n.islands` and defaults to 4. More specifically, under the default settings, the 1000 `n.islands` are split into 250 distinct clusters each containing 4 islands (`island.cluster.size`). Within a cluster, migrations of top chromosomes from one cluster island to another are periodically permitted (controlled by `migration.generations`), and distinct clusters evolve completely independently.
`migration.generations`	An integer equal to the number of generations between migrations among islands of a distinct cluster. Argument `generations` must be an integer multiple of this value. Defaults to 50.
`n.migrations`	The number of chromosomes that migrate among islands. This value must be less than `n.chromosomes` and greater than 0, defaulting to 20.
`recessive.ref.prop`	The proportion to which the observed proportion of informative cases with the provisional risk genotype(s) will be compared to determine whether to recode the SNP as recessive. Defaults to 0.75.
`recode.test.stat`	For a given SNP, the minimum test statistic required to recode and recompute the fitness score using recessive coding. Defaults to 1.64. See the GADGETS paper for specific details.
`dif.coding`	A logical indicating whether, for a given SNP, the case - complement genotype difference should be coded as the sign of the difference (defaulting to false) or the raw difference.

Value

For each island, a list of two elements will be written to `results.dir`:

top.chromosome.results A data.table of the final generation chromosomes, their fitness scores, their difference vectors, and the number of risk alleles required for each chromosome SNP for a case or complement to be classified as having the provisional risk set. See the package vignette for an example and the documentation for `chrom.fitness.score` for additional details.

n.generations The total number of generations run.

Examples

```
data(case)
data(dad)
data(mom)
library(Matrix)
block.ld.mat <- as.matrix(bdiag(list(
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25),
    matrix(rep(TRUE, 25^2), nrow = 25)
)))
pp.list <- preprocess.genetic.data(case[, 1:10],
    father.genetic.data = dad[, 1:10],
    mother.genetic.data = mom[, 1:10],
    block.ld.mat = block.ld.mat[1:10, 1:10]
)
run.gadgets(pp.list,
    n.chromosomes = 4, chromosome.size = 3, results.dir = "tmp",
    cluster.type = "interactive", registryargs = list(
        file.dir = "tmp_reg",
        seed = 1500
    ),
    generations = 2, n.islands = 2, island.cluster.size = 1,
    n.migrations = 0
)

unlink("tmp", recursive = TRUE)
unlink("tmp_reg", recursive = TRUE)
```

`snp.annotations` *RSID, REF, and ALT annotations for example dataset SNPs*

Description

A data.frame containing the RSID, REF allele and ALT allele for each SNP in the example datasets. The SNPs are in the same order as they appear in the example dataset columns.

Usage

```
data(snp.annotations)
```

Format

A data frame with 100 rows and 3 variables

Index

∗ **datasets**
    case, 2
    dad, 10
    mom, 22
    snp.annotations, 33

case, 2
chrom.fitness.score, 3
combine.islands, 6
compute.graphical.scores, 7

dad, 10

epistasis.test, 11
epistasisGA, 13

GADGETS, 13
global.test, 17

mom, 22

network.plot, 23

permute.dataset, 26
preprocess.genetic.data, 27

run.gadgets, 30

snp.annotations, 33