Package 'corehunter'

September 1, 2023

Title Multi-Purpose Core Subset Selection

Version 3.2.3

Date 2023-08-31

Description Core Hunter is a tool to sample diverse, representative subsets from large germplasm collections, with minimum redundancy. Such so-called core collections have applications in plant breeding and genetic resource management in general. Core Hunter can construct cores based on genetic marker data, phenotypic traits or precomputed distance matrices, optimizing one of many provided evaluation measures depending on the precise purpose of the core (e.g. high diversity, representativeness, or allelic richness). In addition, multiple measures can be simultaneously optimized as part of a weighted index to bring the different perspectives closer together.
The Core Hunter library is implemented in Java 8 as an open source project (see <http://www.corehunter.org>).

Depends R (>= 3.2.3), rJava (>= 0.9-8)

Imports naturalsort (>= 0.1.2), methods

SystemRequirements Java (>= 8)

License MIT + file LICENSE

RoxygenNote 7.2.3

Suggests testthat, mockr, StatMatch

Encoding UTF-8

NeedsCompilation no

Author Herman De Beukelaer [aut, cre],
Guy Davenport [aut],
Veerle Fack [ths]

Maintainer Herman De Beukelaer <herman.debeukelaer@gmail.com>

Repository CRAN

Date/Publication 2023-09-01 14:30:02 UTC

R topics documented:

`coreHunterData` *Initialize Core Hunter data.*

Description

The data may contain genotypes, phenotypes and/or a precomputed distance matrix. All provided data should describe the same individuals which is verified by comparing the item ids and names.

Usage

```
coreHunterData(genotypes, phenotypes, distances)
```

Arguments

`genotypes`	Genetic marker data (`chgeno`).
`phenotypes`	Phenotypic trait data (`chpheno`).
`distances`	Precomputed distance matrix (`chdist`).

Value

Core Hunter data (`chdata`) with elements

`geno` Genotype data of class `chgeno` if included.

`pheno` Phenotype data of class `chpheno` if included.

`dist` Distance data of class `chdist` if included.

`size` Number of individuals in the dataset.

`ids` Unique item identifiers.

`names` Item names. Names of individuals to which no explicit name has been assigned are equal to the unique `ids`.

`java` Java version of the data object.

Core Hunter data of class `chdata`.

See Also

genotypes, phenotypes, distances

Examples

```
## Not run:
geno.file <- system.file("extdata", "genotypes.csv", package = "corehunter")
pheno.file <- system.file("extdata", "phenotypes.csv", package = "corehunter")
dist.file <- system.file("extdata", "distances.csv", package = "corehunter")

my.data <- coreHunterData(
  genotypes(file = geno.file, format = "default"),
  phenotypes(file = pheno.file),
  distances(file = dist.file)
)

## End(Not run)
```

---

distances *Create Core Hunter distance data from matrix or file.*

Description

Specify either a symmetric distance matrix or the file from which to read the matrix. See https://www.corehunter.org for documentation and examples of the distance matrix file format used by Core Hunter.

Usage

```
distances(data, file)
```

Arguments

data	Symmetric distance matrix. Unique row and column headers are required, should be the same and are used as item ids. Can be a numeric matrix or a data frame. The data frame may optionally include a first column NAME used to assign names to some or all individuals. The remaining columns should be numeric.
file	File from which to read the distance matrix.

Value

Distance matrix data of class chdist with elements

data Distance matrix (numeric matrix).

size Number of individuals in the dataset.

ids Unique item identifiers.

`names` Item names. Names of individuals to which no explicit name has been assigned are equal to the unique `ids`.

`java` Java version of the data object.

`file` Normalized path of file from which data was read (if applicable).

Examples

```
# create from distance matrix
m <- matrix(runif(100), nrow = 10, ncol = 10)
diag(m) <- 0
# make symmetric
m[lower.tri(m)] <- t(m)[lower.tri(m)]
# set headers
rownames(m) <- colnames(m) <- paste("i", 1:10, sep = "-")

dist <- distances(m)

# read from file
dist.file <- system.file("extdata", "distances.csv", package = "corehunter")
dist <- distances(file = dist.file)
```

---

evaluateCore *Evaluate a core collection using the specified objective.*

Description

Evaluate a core collection using the specified objective.

Usage

```
evaluateCore(core, data, objective)
```

Arguments

`core`	A core collection of class `chcore`, or a numeric or character vector indicating the indices or ids, respectively, of the individuals in the evaluated core.
`data`	Core Hunter data (`chdata`) containing genotypes, phenotypes and/or a precomputed distance matrix. Can also be an object of class `chdist`, `chgeno` or `chpheno` if only one type of data is provided.
`objective`	Objective function (`chobj`) used to evaluate the core.

Value

Value of the core when evaluated with the chosen objective (numeric).

See Also

`coreHunterData`, `objective`

Examples

```
data <- exampleData()
core <- sampleCore(data, objective("EN", "PD"))
evaluateCore(core, data, objective("EN", "PD"))
evaluateCore(core, data, objective("AN", "MR"))
evaluateCore(core, data, objective("EE", "GD"))
evaluateCore(core, data, objective("CV"))
evaluateCore(core, data, objective("HE"))
```

`exampleData` *Small example dataset with 218 individuals.*

Description

Data was genotyped using 190 SNP markers and 4 quantitative traits were recorded. Includes a pre-computed distance matrix read from `"extdata/distances.csv"`, genotypes read from `"extdata/genotypes-biparental.` and phenotypes read from `"extdata/phenotypes.csv"`. The distance matrix is computed from the genotypes (Modified Rogers' distance).

Usage

```
exampleData()
```

Details

Data was taken from the CIMMYT Research Data Repository (Study Global ID hdl:11529/10199; real data set 5, cycle 0).

Value

Core Hunter data of class `chdata`

Source

Cerón-Rojas, J. Jesús ; Crossa, José; Arief, Vivi N.; Kaye Basford; Rutkoski, Jessica; Jarquín, Diego ; Alvarado, Gregorio; Beyene, Yoseph; Semagn, Kassa ; DeLacy, Ian, 2015-06-04, "Application of a Genomics Selection Index to Real and Simulated Data", http://hdl.handle.net/11529/10199 V10

Examples

```
exampleData()
```

genotypes *Create Core Hunter genotype data from data frame, matrix or file.*

Description

Specify either a data frame or matrix, or a file from which to read the genotypes. See https://www.corehunter.org for documentation and examples of the genotype data file format used by Core Hunter.

Usage

```
genotypes(data, alleles, file, format)
```

Arguments

data	Data frame or matrix containing the genotypes (individuals x markers) depending on the chosen format: `default` Data frame. One row per individual and one or more columns per marker. Columns contain the names, numbers, references, ... of observed alleles. Unique row names (item ids) are required and columns should be named after the marker to which they belong, optionally extended with an arbitrary suffix starting with a dot (.), dash (-) or underscore (_) character. `biparental` Numeric matrix or data frame. One row per individual and one column per marker. Data consists of 0, 1 and 2 coding for homozygous (AA), heterozygous (AB) and homozygous (BB), respectively. Unique row names (item ids) are required and optionally column (marker) names may be included as well. `frequency` Numeric matrix or data frame. One row per individual (or bulk sample) and multiple columns per marker. Data consists of allele frequencies, grouped per marker in consecutive columns named after the corresponding marker, optionally extended with an arbitrary suffix starting with a dot (.), dash (-) or underscore (_) character.. The allele frequencies of each marker should sum to one in each sample. Unique row names (item ids) are required. In case a data frame is provided, an optional first column `NAME` may be included to specify item names. The remaining columns should follow the format as described above. See https://www.corehunter.org for more details about the supported genotype formats. Note that both the `frequency` and `biparental` format syntactically also comply with the `default` format but with different semantics, meaning that it is very important to specify the correct format. Some checks have been built in that raise warnings in case it seems that the wrong format might have been specified based on an inspection of the data. If you are sure that you have selected the correct format these warnings, if any, can be safely ignored.
alleles	Allele names per marker (`character` vector). Ignored except when creating `frequency` data from a matrix or data frame. Allele names should be ordered in correspondence with the data columns.

file	File containing the genotype data.
format	Genotype data format, one of default, biparental or frequency.

Value

Genotype data of class chgeno with elements

data Genotypes. Data frame for default format, numeric matrix for other formats.

size Number of individuals in the dataset.

ids Unique item identifiers (character).

names Item names (character). Names of individuals to which no explicit name has been assigned are equal to the unique ids.

markers Marker names (character). May contain NA values in case only some or no marker names were specified. Marker names are always included for the default and frequency format but are optional for the biparental format.

alleles List of character vectors with allele names per marker. Vectors may contain NA values in case only some or no allele names were specified. For biparental data the two alleles are name "0" and "1", respectively, for all markers. For the default format allele names are inferred from the provided data. Finally, for frequency data allele names are optional and may be specified either in the file or through the alleles argument when creating this type of data from a matrix or data frame.

java Java version of the data object.

format Genotype data format used.

file Normalized path of file from which data was read (if applicable).

Examples

```
## Not run:
# create from data frame or matrix

# default format
geno.data <- data.frame(
 NAME = c("Alice", "Bob", "Carol", "Dave", "Eve"),
 M1.1 = c(1,2,1,2,1),
 M1.2 = c(3,2,2,3,1),
 M2.1 = c("B","C","D","B",NA),
 M2.2 = c("B","A","D","B",NA),
 M3.1 = c("a1","a1","a2","a2","a1"),
 M3.2 = c("a1","a2","a2","a1","a1"),
 M4.1 = c(NA,"+","+","+","-"),
 M4.2 = c(NA,"-","+","-","-"),
 row.names = paste("g", 1:5, sep = "-")
)
geno <- genotypes(geno.data, format = "default")

# biparental (e.g. SNP)
geno.data <- matrix(
 sample(c(0,1,2), replace = TRUE, size = 1000),
```

```
 nrow = 10, ncol = 100
)
rownames(geno.data) <- paste("g", 1:10, sep = "-")
colnames(geno.data) <- paste("m", 1:100, sep = "-")
geno <- genotypes(geno.data, format = "biparental")

# frequencies
geno.data <- matrix(
 c(0.0, 0.3, 0.7, 0.5, 0.5, 0.0, 1.0,
   0.4, 0.0, 0.6, 0.1, 0.9, 0.0, 1.0,
   0.3, 0.3, 0.4, 1.0, 0.0, 0.6, 0.4),
 byrow = TRUE, nrow = 3, ncol = 7
)
rownames(geno.data) <- paste("g", 1:3, sep = "-")
colnames(geno.data) <- c("M1", "M1", "M1", "M2", "M2", "M3", "M3")
alleles <- c("M1-a", "M1-b", "M1-c", "M2-a", "M2-b", "M3-a", "M3-b")
geno <- genotypes(geno.data, alleles, format = "frequency")

# read from file

# default format
geno.file <- system.file("extdata", "genotypes.csv", package = "corehunter")
geno <- genotypes(file = geno.file, format = "default")

# biparental (e.g. SNP)
geno.file <- system.file("extdata", "genotypes-biparental.csv", package = "corehunter")
geno <- genotypes(file = geno.file, format = "biparental")

# frequencies
geno.file <- system.file("extdata", "genotypes-frequency.csv", package = "corehunter")
geno <- genotypes(file = geno.file, format = "frequency")

## End(Not run)
```

`getAlleleFrequencies` *Get Allele frequency matrix.*

Description

Get Allele frequency matrix.

Usage

```
getAlleleFrequencies(data)
```

Arguments

`data`	Core Hunter data containing genotypes

Value

allele frequency matrix

`getNormalizationRanges` *Determine normalization ranges of all objectives in a multi-objective configuration.*

Description

Executes an independent stochastic hill-climbing search (random descent) per objective to approximate the optimal solution for each objective, from which a suitable normalization range is inferred based on the Pareto minima/maxima. These normalization searches are executed in parallel.

Usage

```
getNormalizationRanges(
  data,
  obj,
  size = 0.2,
  always.selected = integer(0),
  never.selected = integer(0),
  mode = c("default", "fast"),
  time = NA,
  impr.time = NA,
  steps = NA,
  impr.steps = NA
)
```

Arguments

`data`	Core Hunter data (`chdata`) containing genotypes, phenotypes and/or a precomputed distance matrix. Can also be an object of class `chdist`, `chgeno` or `chpheno` if only one type of data is provided.
`obj`	List of objectives (`chobj`). If no objectives are specified Core Hunter maximizes a weighted index including the default entry-to-nearest-entry distance (`EN`) for each available data type. For genotypes, the Modified Roger's distance (`MR`) is used. For phenotypes, Gower's distance (`GD`) is applied.
`size`	Desired core subset size (numeric). If larger than one the value is used as the absolute core size after rounding. Else it is used as the sampling rate and multiplied with the dataset size to determine the size of the core. The default sampling rate is 0.2.
`always.selected`	vector with indices (integer) or ids (character) of items that should always be selected in the core collection

never.selected vector with indices (integer) or ids (character) of items that should never be selected in the core collection

mode Execution mode (default or fast). In default mode, the normalization searches terminate when no improvement is found for ten seconds. In fast mode, searches terminate as soon as no improvement is made for two seconds. These stop conditions can be overridden using arguments time, impr.time, steps and/or impr.steps. In default mode, the value of the latter two, step-based conditions is multiplied with 500, in line with the behaviour of sampleCore when executed in default mode.

time Absolute runtime limit in seconds. Not used by default (NA). If used, it should be a strictly positive value, which is rounded to the nearest integer.

impr.time Maximum time without improvement in seconds. If no explicit stop conditions are specified, the maximum time without improvement defaults to ten or two seconds, when executing Core Hunter in default or fast mode, respectively. If a custom improvement time is specified, it should be strictly positive and is rounded to the nearest integer.

steps Maximum number of search steps. Not used by default (NA). If used, it should be a strictly positive value, which is rounded to the nearest integer. In default mode, the value is multiplied with 500, in line with the behaviour of sampleCore when executed in default mode.

impr.steps Maximum number of steps without improvement. Not used by default (NA). If used, it should be a strictly positive value, which is rounded to the nearest integer. In default mode, the value is multiplied with 500, in line with the behaviour of sampleCore when executed in default mode.

Details

For an objective that is being maximized, the upper bound is set to the value of the best solution for that objective, while the lower bound is set to the Pareto minimum, i.e. the minimum value obtained when evaluating all optimal solutions (for each single objective) with the considered objective. For an objective that is being minimized, the roles of upper and lower bound are interchanged, and the Pareto maximum is used instead.

Because Core Hunter uses stochastic algorithms, repeated runs may produce different results. To eliminate randomness, you may set a random number generation seed using set.seed prior to executing Core Hunter. In addition, when reproducible results are desired, it is advised to use step-based stop conditions instead of the (default) time-based criteria, because runtimes may be affected by external factors, and, therefore, a different number of steps may have been performed in repeated runs when using time-based stop conditions.

Value

Numeric matrix with one row per objective and two columns:

lower Lower bound of normalization range.

upper Upper bound of normalization range.

See Also

coreHunterData, objective

Examples

```
data <- exampleData()

# maximize entry-to-nearest-entry distance between genotypes and phenotypes (equal weight)
objectives <- list(objective("EN", "MR"), objective("EN", "GD"))
# get normalization ranges for default size (20%)
ranges <- getNormalizationRanges(data, obj = objectives, mode = "fast")

# set normalization ranges and sample core
objectives <- lapply(1:2, function(o){setRange(objectives[[o]], ranges[o,])})
core <- sampleCore(data, obj = objectives)
```

`objective` *Create Core Hunter objective.*

Description

The following optimization objectives are supported by Core Hunter:

`EN` Average entry-to-nearest-entry distance (default). Maximizes the average distance between each selected individual and the closest other selected item in the core. Favors diverse cores in which each individual is sufficiently different from the most similar other selected item (low redundancy). Multiple distance measures are provided to be used with this objective (see below).

`AN` Average accession-to-nearest-entry distance. Minimizes the average distance between each individual (from the full dataset) and the closest selected item in the core (which can be the individual itself). Favors representative cores in which all items from the original dataset are represented by similar individuals in the selected subset. Multiple distance measures are provided to be used with this objective (see below).

`EE` Average entry-to-entry distance. Maximizes the average distance between each pair of selected individuals in the core. This objective is related to the entry-to-nearest-entry (EN) distance but less effectively avoids redundant, similar individuals in the core. In general, use of `EN` is preferred. Multiple distance measures are provided to be used with this objective (see below).

`SH` Shannon's allelic diversity index. Maximizes the entropy, as used in information theory, of the selected core. Independently takes into account all allele frequencies, regardless of the locus (marker) where to which the allele belongs. Requires genotypes.

`HE` Expected proportion of heterozygous loci. Maximizes the expected proportion of heterozygous loci in offspring produced from random crossings within the selected core. In contrast to Shannon's index (`SH`) this objective treats each marker (locus) with equal importance, regardless of the number of possible alleles for that marker. Requires genotypes.

CV Allele coverage. Maximizes the proportion of alleles observed in the full dataset that are retained in the selected core. Requires genotypes.

The first three objective types (EN, AN and EE) aggregate pairwise distances between individuals. These distances can be computed using various measures:

MR Modified Rogers distance (default). Requires genotypes.

CE Cavalli-Sforza and Edwards distance. Requires genotypes.

GD Gower distance. Requires phenotypes.

PD Precomputed distances. Uses the precomputed distance matrix of the dataset.

Usage

```
objective(
  type = c("EN", "AN", "EE", "SH", "HE", "CV"),
  measure = c("MR", "CE", "GD", "PD"),
  weight = 1,
  range = NULL
)
```

Arguments

type	Objective type, one of EN (default), AN, EE, SH, HE or CV (see description). The former three objectives are distance based and require to choose a distance measure. By default, Modified Roger's distance is used, computed from the genotypes.
measure	Distance measure used to compute the distance between two individuals, one of MR (default), CE, GD or PD (see description). Ignored when type is SH, HE or CV.
weight	Weight assigned to the objective when maximizing a weighted index. Defaults to 1.0.
range	Normalization range [l,u] of the objective when maximizing a weighted index. By default the range is not set (NULL) and will be determined automatically prior to execution, if normalization is enabled (default). Values are rescaled to [0,1] with the linear formula $v' = (v - l)/(u - l)$. When an explicit normalization range is set, it overrides the automatically inferred range. Also, setting the range for all included objectives reduces the computation time when sampling a multi-objective core collection. In case of repeated sampling from the same dataset with the same objectives and size, it is therefore advised to determine the normalization ranges only once using getNormalizationRanges so that they can be reused for all executions.

Value

Core Hunter objective of class chobj with elements

type Objective type.

meas Distance measure (if applicable).

weight Assigned weight.

range Normalization range (if specified).

See Also

getNormalizationRanges, setRange

Examples

```
objective()
objective(meas = "PD")
objective("EE", "GD")
objective("HE")
objective("EN", "MR", range = c(0.150, 0.300))
objective("AN", "MR", weight = 0.5, range = c(0.150, 0.300))
```

---

phenotypes *Create Core Hunter phenotype data from data frame or file.*

Description

Specify either a data frame containing the phenotypic trait observations or a file from which to read the data. See https://www.corehunter.org for documentation and examples of the phenotype data format used by Core Hunter.

Usage

```
phenotypes(data, types, min, max, file)
```

Arguments

data	Data frame containing one row per individual and one column per trait. Unique row and column names are required and used as item and trait ids, respectively. The data frame may optionally include a first column NAME used to assign names to some or all individuals.
types	Variable types (optional). Vector of characters, each of length one or two. Ignored when reading from file. The first letter indicates the scale type and should be one of N (nominal), O (ordinal), I (interval) or R (ratio). The second letter optionally indicates the variable encoding (in Java) and should be one of B (boolean), T (short), I (integer), L (long), R (big integer), F (float), D (double), M (big decimal), A (date) or S (string). The default encoding is S (string) for nominal variables, I (integer) for ordinal and interval variables and D (double) for ratio variables. Interval and ratio variables are limited to numeric encodings. If no explicit variable types are specified these are automatically inferred from the data frame column types and classes, whenever possible. Columns of type character are treated as nominal string encoded variables (N). Unordered factor columns are converted to character and also treated as string encoded nominals. Ordered factors are converted to integer encoded interval variables (I) as

described below. Columns of type `logical` are taken to be asymmetric binary variables (`NB`). Finally, `integer` and more broadly `numeric` columns are treated as integer encoded interval variables (`I`) and double encoded ratio variables (`R`), respectively.

Boolean encoded nominals (`NB`) are treated as asymmetric binary variables. For symmetric binary variables just use the default string encoding (`N` or `NS`). Other nominal variables are converted to factors.

Ordinal variables of class `ordered` are converted to integers respecting the order and range of the factor levels and subsequently treated as integer encoded interval variables (`I`). This conversion allows to model the full range of factor levels also when some might not occur in the data. For other ordinal variables it is assumed that each value occurs at least once and that values follow the natural ordering of the chosen data type (in Java).

If explicit types are given for some variables others can still be automatically inferred by setting their type to `NA`.

`min` Minimum values of interval or ratio variables (optional). Numeric vector. Ignored when reading from file. If undefined for some variables the respective minimum is inferred from the data. If the data exceeds the minimum it is also updated accordingly. For nominal and ordinal variables just put `NA`.

`max` Maximum values of interval or ratio variables (optional). Numeric vector. Ignored when reading from file. If undefined for some variables the respective maximum is inferred from the data. If the data exceeds the maximum it is also updated accordingly. For nominal and ordinal variables just put `NA`.

`file` File containing the phenotype data.

Value

Phenotype data of class `chpheno` with elements

`data` Phenotypes (data frame).

`size` Number of individuals in the dataset.

`ids` Unique item identifiers.

`names` Item names. Names of individuals to which no explicit name has been assigned are equal to the unique `ids`.

`types` Variable types and encodings.

`ranges` Variable ranges, when applicable (`NA` elsewhere).

`java` Java version of the data object.

`file` Normalized path of file from which the data was read (if applicable).

Examples

```
# create from data frame
pheno.data <- data.frame(
 season = c("winter", "summer", "summer", "winter", "summer"),
 yield = c(34.5, 32.6, 22.1, 54.12, 43.33),
 size = ordered(c("l", "s", "s", "m", "l"), levels = c("s", "m", "l")),
```

```
 resistant = c(FALSE, TRUE, TRUE, FALSE, TRUE)
)
pheno <- phenotypes(pheno.data)

# explicit types
pheno <- phenotypes(pheno.data, types = c("N", "R", "O", "NB"))
# treat last column as symmetric binary, auto infer others
pheno <- phenotypes(pheno.data, types = c(NA, NA, NA, "NS"))

# explicit ranges
pheno <- phenotypes(pheno.data, min = c(NA, 20.0, NA, NA), max = c(NA, 60.0, NA, NA))

# read from file
pheno.file <- system.file("extdata", "phenotypes.csv", package = "corehunter")
pheno <- phenotypes(file = pheno.file)
```

`read.autodelim` *Read delimited file.*

Description

Delegates to `read.delim` where the separator is inferred from the file extension (CSV or TXT). For CSV files the delimiter is set to `","` while for TXT file `"\t"` is used. Also sets some default argument values as used by Core Hunter.

Usage

```
read.autodelim(
  file,
  quote = "'\"",
  row.names = 1,
  na.strings = "",
  check.names = FALSE,
  strip.white = TRUE,
  stringsAsFactors = FALSE,
  ...
)
```

Arguments

`file`	File path.
`quote`	the set of quoting characters. To disable quoting altogether, use `quote = ""`. See `scan` for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless `colClasses` is specified.

`row.names` a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names.

If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if `row.names` is missing, the rows are numbered.

Using `row.names = NULL` forces row numbering. Missing or `NULL` `row.names` generate row names that are considered to be 'automatic' (and not preserved by `as.matrix`).

`na.strings` a character vector of strings which are to be interpreted as `NA` values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens *after* white space is stripped from the input, so `na.strings` values may need their own white space stripped in advance.

`check.names` logical. If `TRUE` then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by `make.names`) so that they are, and also to ensure that there are no duplicates.

`strip.white` logical. Used only when `sep` has been specified, and allows the stripping of leading and trailing white space from unquoted `character` fields (`numeric` fields are always stripped). See `scan` for further details (including the exact meaning of 'white space'), remembering that the columns may include the row names.

`stringsAsFactors` logical: should character vectors be converted to factors? Note that this is overridden by `as.is` and `colClasses`, both of which allow finer control.

`...` Further arguments to be passed to `read.delim`.

Value

Data frame.

`sampleCore` *Sample a core collection.*

Description

Sample a core collection from the given data.

Usage

```
sampleCore(
  data,
  obj,
  size = 0.2,
  always.selected = integer(0),
```

```
  never.selected = integer(0),
  mode = c("default", "fast"),
  normalize = TRUE,
  time = NA,
  impr.time = NA,
  steps = NA,
  impr.steps = NA,
  indices = FALSE,
  verbose = FALSE
)
```

Arguments

data	Core Hunter data (chdata) containing genotypes, phenotypes and/or a precomputed distance matrix. Typically the data is obtained with coreHunterData. Can also be an object of class chdist, chgeno or chpheno if only one type of data is provided.
obj	Objective or list of objectives (chobj). If no objectives are specified Core Hunter maximizes a weighted index including the default entry-to-nearest-entry distance (EN) for each available data type, with equal weight. For genotypes, the Modified Roger's distance (MR) is used. For phenotypes, Gower's distance (GD) is applied.
size	Desired core subset size (numeric). If larger than one the value is used as the absolute core size after rounding. Else it is used as the sampling rate and multiplied with the dataset size to determine the size of the core. The default sampling rate is 0.2.
always.selected	vector with indices (integer) or ids (character) of items that should always be selected in the core collection
never.selected	vector with indices (integer) or ids (character) of items that should never be selected in the core collection
mode	Execution mode (default or fast). In default mode, Core Hunter uses an advanced parallel tempering search algorithm and terminates when no improvement is found for ten seconds. In fast mode, a simple stochastic hill-climbing algorithm is applied and Core Hunter terminates as soon as no improvement is made for two seconds. Stop conditions can be overridden with arguments time and impr.time.
normalize	If TRUE (default), the applied objectives in a multi-objective configuration (two or more objectives) are automatically normalized prior to execution. For single-objective configurations, this argument is ignored. Normalization requires an independent preliminary search per objective (fast stochastic hill-climber, executed in parallel for all objectives). The same stop conditions, as specified for the main search, are also applied to each normalization search. In default execution mode, however, any step-based stop conditions are multiplied by 500 for the normalization searches, because in that case the main search (parallel tempering) executes 500 stochastic hill-climbing steps per replica, in a single step of the main search.

Normalization ranges can also be precomputed (see getNormalizationRanges) or manually specified in the objectives to save computation time when sampling core collections. This is especially useful when multiple cores are sampled for the same objectives, with possibly varying weights.

`time` Absolute runtime limit in seconds. Not used by default (NA). If used, it should be a strictly positive value, which is rounded to the nearest integer.

`impr.time` Maximum time without improvement in seconds. If no explicit stop conditions are specified, the maximum time without improvement defaults to ten or two seconds, when executing Core Hunter in `default` or `fast` mode, respectively. If a custom improvement time is specified, it should be strictly positive and is rounded to the nearest integer.

`steps` Maximum number of search steps. Not used by default (NA). If used, it should be a strictly positive value, which is rounded to the nearest integer. The number of steps applies to the main search. Details of how this stop condition is transferred to normalization searches, in a multi-objective configuration, are provided in the description of the argument `normalize`.

`impr.steps` Maximum number of steps without improvement. Not used by default (NA). If used, it should be a strictly positive value, which is rounded to the nearest integer. The maximum number of steps without improvement applies to the main search. Details of how this stop condition is transferred to normalization searches, in a multi-objective configuration, are provided in the description of the argument `normalize`.

`indices` If TRUE, the result contains the indices instead of ids (default) of the selected individuals.

`verbose` If TRUE, search progress messages are printed to the console. Defaults to FALSE.

Details

Because Core Hunter uses stochastic algorithms, repeated runs may produce different results. To eliminate randomness, you may set a random number generation seed using `set.seed` prior to executing Core Hunter. In addition, when reproducible results are desired, it is advised to use step-based stop conditions instead of the (default) time-based criteria, because runtimes may be affected by external factors, and, therefore, a different number of steps may have been performed in repeated runs when using time-based stop conditions.

Value

Core subset (`chcore`). It has an element `sel` which is a character or numeric vector containing the sorted ids or indices, respectively, of the selected individuals (see argument `indices`). In addition the result has one or more elements that indicate the value of each objective function that was included in the optimization.

See Also

coreHunterData, objective, getNormalizationRanges

Examples

```
data <- exampleData()

# default size, maximize entry-to-nearest-entry Modified Rogers distance
obj <- objective("EN", "MR")
core <- sampleCore(data, obj)

# fast mode
core <- sampleCore(data, obj, mode = "f")
# absolute size
core <- sampleCore(data, obj, size = 25)
# relative size
core <- sampleCore(data, obj, size = 0.1)

# other objective: minimize accession-to-nearest-entry precomputed distance
core <- sampleCore(data, obj = objective(type = "AN", measure = "PD"))
# multiple objectives (equal weight)
core <- sampleCore(data, obj = list(
 objective("EN", "PD"),
 objective("AN", "GD")
))
# multiple objectives (custom weight)
core <- sampleCore(data, obj = list(
 objective("EN", "PD", weight = 0.3),
 objective("AN", "GD", weight = 0.7)
))

# custom stop conditions
core <- sampleCore(data, obj, time = 5, impr.time = 2)
core <- sampleCore(data, obj, steps = 300)

# print progress messages
core <- sampleCore(data, obj, verbose = TRUE)
```

`setRange` *Set the normalization range of the given objective.*

Description

See argument `range` of `objective` for details.

Usage

```
setRange(obj, range)
```

Arguments

obj	Core Hunter objective of class chobj.
range	Normalization range [l,u]. See argument range of objective for details.

Value

Objective including normalization range.

See Also

objective

---

wrapData *Wrap distances, genotypes or phenotypes in Core Hunter data.*

Description

If the given data does not match any of these three classes it is returned unchanged.

Usage

```
wrapData(data)
```

Arguments

data	of class chgeno, chpheno or chdist

Value

Core Hunter data of class chdata

Index