Package ‘hmsr’

October 25, 2023

Title Multipopulation Evolutionary Strategy HMS

Version 1.0.1

Description The HMS (Hierarchic Memetic Strategy) is a composite global optimization strategy consisting of a multi-population evolutionary strategy and some auxiliary methods. The HMS makes use of a dynamically-evolving data structure that provides an organization among the component populations. It is a tree with a fixed maximal height and variable internal node degree. Each component population is governed by a particular evolutionary engine. This package provides a simple R implementation with examples of using different genetic algorithms as the population engines. References: J. Sawicki, M. Łoś, M. Smołka, J. Alvarez-Aramberri (2022) <doi:10.1007/s11047-020-09836-w>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

Imports GA, msm, methods, uuid, graphics

Suggests testthat (>= 3.0.0), ecr, filelock, parallel, doParallel, grDevices, smoof

BugReports `https://github.com/WojtAcht/hms/issues`

URL `https://wojtacht.github.io/hms/`

NeedsCompilation no

Author Wojciech Achtelik [aut, cre],
Marcin Kozubek [aut],
Maciej Smołka [ths, aut] (<`https://orcid.org/0000-0002-3386-0555`>, Java original),
AGH University of Kraków [cph]

Maintainer Wojciech Achtelik <`wachtelik@gmail.com`>

Repository CRAN

Date/Publication 2023-10-25 07:10:02 UTC

R topics documented:

`default_run_gradient_method` *Function that runs gradient method for one deme. Wrapper function for stats::optim.*

Description

Function that runs gradient method for one deme. Wrapper function for stats::optim.

Usage

```
default_run_gradient_method(deme, fitness, optim_args)
```

Arguments

`deme`	• Deme
`fitness`	• fitness function
`optim_args`	• list of additional parameters (stats::optim parameters)

Value

list with named fields: solution, population, value. See `ga_metaepoch` for more details.

`ecr_metaepoch` *Function that runs one ecr metaepoch. Wrapper function for ecr::ecr.*

Description

Function that runs one ecr metaepoch. Wrapper function for ecr::ecr.

Usage

```
ecr_metaepoch(config_ecr)
```

Arguments

`config_ecr`	• list of ecr::ecr params

Value

list with named fields: solution, population, value. See `ga_metaepoch` for more details.

Examples

```
tree_height <- 3
empty_config_ecr <- lapply(1:tree_height, function(x) {
  list()
})
ecr_metaepoch(empty_config_ecr)
```

euclidean_distance *Euclidean distance*

Description

Euclidean distance

Usage

```
euclidean_distance(x, y)
```

Arguments

x	• numeric
y	• numeric

Value

numeric - euclidean distance between x and y

Examples

```
euclidean_distance(c(1, 1), c(1, 2))
```

ga_metaepoch *Function that runs one GA metaepoch. Wrapper function for GA::ga.*

Description

Function that runs one GA metaepoch. Wrapper function for GA::ga.

Usage

```
ga_metaepoch(config_ga)
```

Arguments

config_ga	• list of GA::ga params

Value

list with named fields: solution, population, value. A solution is a value of the decision variable giving the best fitness. A population is a matrix representing final population. Value is the value of a fitness function for the solution.

Examples

```
tree_height <- 3
empty_config_ga <- lapply(1:tree_height, function(x) {
  list()
})
ga_metaepoch(empty_config_ga)
```

`gsc_max_fitness_evaluations`

Factory function for a global stopping condition that stops the computation after fitness function has been evaluated given number of times.

Description

Factory function for a global stopping condition that stops the computation after fitness function has been evaluated given number of times.

Usage

```
gsc_max_fitness_evaluations(max_evaluations)
```

Arguments

`max_evaluations`

- numeric - maximum number of fitness function evaluations

Value

Function that receives a list of metaepoch snapshots and returns a Boolean value determining whether the computation should be stopped based on how many fitness function evaluations have been made, which can be used as a global stopping condition for the hms function.

Examples

```
global_stopping_condition <- gsc_max_fitness_evaluations(10000)
```

`gsc_metaepochs_count` *Factory function for a global stopping condition that stops the computation after given number of metaepochs.*

Description

Factory function for a global stopping condition that stops the computation after given number of metaepochs.

Usage

```
gsc_metaepochs_count(metaepochs_count)
```

Arguments

`metaepochs_count`	• numeric - maximum number of metaepochs

Value

Function that receives a list of metaepoch snapshots and returns a Boolean value determining whether the computation should be stopped based on how many metaepochs have passed, which can be used as a global stopping condition for the hms function.

Examples

```
global_stopping_condition <- gsc_metaepochs_count(10)
```

`gsc_trivial` *Factory function for a global stopping condition that never stops the computation. It results in hms running until there are no more active demes.*

Description

Factory function for a global stopping condition that never stops the computation. It results in hms running until there are no more active demes.

Usage

```
gsc_trivial()
```

Value

function that always returns `FALSE`, which can be used as a global stopping condition for the hms function.

Examples

```
global_stopping_condition <- gsc_trivial()
```

---

hms — *Maximization (or minimization) of a fitness function using Hierarchic Memetic Strategy.*

Description

Maximization (or minimization) of a fitness function using Hierarchic Memetic Strategy.

Usage

```
hms(
  tree_height = 3,
  minimize = FALSE,
  fitness,
  lower,
  upper,
  sigma = default_sigma(lower, upper, tree_height),
  population_sizes = default_population_sizes(tree_height),
  run_metaepoch = default_ga_metaepoch(tree_height),
  gsc = gsc_default,
  lsc = lsc_default,
 sc = sc_max_metric(euclidean_distance, sprouting_default_euclidean_distances(sigma)),
  create_population = default_create_population(sigma),
  suggestions = NULL,
  with_gradient_method = FALSE,
  gradient_method_args = default_gradient_method_args,
  run_gradient_method,
  monitor_level = "basic",
  parallel = FALSE
)
```

Arguments

`tree_height`	numeric - default value: 5. It determines the maximum tree height which will usually be reached unless a very strict local stopping condition, global stopping condition or sprouting condition is used.
`minimize`	logical - `TRUE` when fitness shall be minimized.
`fitness`	fitness function, that returns a numerical value, to be optimized by the strategy.
`lower`	numeric - lower bound of the domain, a vector of length equal to the decision variables.
`upper`	numeric - upper bound of the domain, a vector of length equal to the decision variables.

`sigma`	numeric - Vector of standard deviations for each tree level used to create a population of a sprouted deme.
`population_sizes`	numeric - Sizes of deme populations on each tree level.
`run_metaepoch`	A function that takes 5 parameters: fitness, suggestions, lower, upper, tree_level, runs a metaepoch on the given deme population and returns list with 3 named fields: solution, population, value.
`gsc`	global stopping condition function taking a list of MetaepochSnapshot objects and returning a logical value; it is evaluated after every metaepoch and determines whether whole computation should be stopped. See `gsc_metaepochs_count` for more details.
`lsc`	local stopping condition - function taking a deme and a list of MetaepochSnapshot objects representing previous metaepochs; it is run on every deme after it has run a metaepoch and determines whether that deme will remain active. See `lsc_max_fitness_evaluations` for more details.
`sc`	sprouting condition - function taking 3 arguments: an individual, a tree level and a list of Deme objects; it determines whether the given individual can sprout a new deme on the given level. See `sc_max_metric` for more details.
`create_population`	function taking 6 parameters: mean, lower, upper, population_size, tree_level, sigma that returns a population for a Deme object to be created on the given tree level.
`suggestions`	matrix of individuals for the initial population of the root
`with_gradient_method`	logical determining whether a gradient method should be run for all leaves at the end of the computation to refine their best solutions.
`gradient_method_args`	list of parameters that are passed to the gradient method
`run_gradient_method`	function - returns list with named fields: solution, population, value
`monitor_level`	string - one of: 'none', 'basic', 'basic_tree', 'verbose_tree'.
`parallel`	logical - `TRUE` when run_metaepoch runs in parallel.

Value

Returns an object of class hms.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
```

`hms-class` *A S4 class representing a result of hms.*

Description

A S4 class representing a result of hms.

Slots

`root_id` character - UUID of a root Deme.

`metaepoch_snapshots` list of objects of class MetaepochSnapshot.

`best_fitness` numeric - best fitness value of all metaepochs.

`best_solution` numeric - best solution of all metaepochs.

`total_time_in_seconds` numeric - time of a hms execution in seconds.

`total_metaepoch_time_in_seconds` numeric - time of all metaepochs in seconds.

`metaepochs_count` numeric - total number of all metaepochs.

`deme_population_sizes` numeric - sizes of deme populations on each tree level. Same as `population_sizes` parameter of hms function.

`lower` numeric - lower bound of the domain, a vector of length equal to the decision variables.

`upper` numeric - upper bound of the domain, a vector of length equal to the decision variables.

`call` language - an object of class "call" representing the matched call.

`lsc_max_fitness_evaluations` *Factory function for a local stopping condition that stops a deme after given number of fitness function evaluations has been made in that deme.*

Description

Factory function for a local stopping condition that stops a deme after given number of fitness function evaluations has been made in that deme.

Usage

```
lsc_max_fitness_evaluations(max_evaluations)
```

Arguments

`max_evaluations`

- numeric

Value

Function that can be used as a local stopping condition for hms.

Examples

```
local_stopping_condition <- lsc_max_fitness_evaluations(500)
```

`lsc_metaepochs_without_active_child`
Factory function for a local stopping condition that stops a deme after given number of metaepochs have past since last metaepoch during which this deme had an active child.

Description

Factory function for a local stopping condition that stops a deme after given number of metaepochs have past since last metaepoch during which this deme had an active child.

Usage

```
lsc_metaepochs_without_active_child(metaepochs_limit)
```

Arguments

`metaepochs_limit`
- number of metaepochs that a deme can be active without any active child

Value

Function that can be used as a local stopping condition for hms.

Examples

```
local_stopping_condition <- lsc_metaepochs_without_active_child(3)
```

`lsc_metaepochs_without_improvement`
Factory function for a local stopping condition that stops a deme after given number of consecutive metaeopochs without an improvement of the best solution found in that deme.

Description

Factory function for a local stopping condition that stops a deme after given number of consecutive metaeopochs without an improvement of the best solution found in that deme.

Usage

```
lsc_metaepochs_without_improvement(max_metaepochs_without_improvement)
```

Arguments

`max_metaepochs_without_improvement`	• numeric

Value

Function that can be used as a local stopping condition for hms.

Examples

```
local_stopping_condition <- lsc_metaepochs_without_improvement(5)
```

`lsc_trivial` *Factory function for a trivial local stopping condition that lets a deme be active forever. It is usually used in the root of a hms tree.*

Description

Factory function for a trivial local stopping condition that lets a deme be active forever. It is usually used in the root of a hms tree.

Usage

```
lsc_trivial()
```

Value

Function that always returns `FALSE`, which can be used as a local stopping condition for hms.

Examples

```
local_stopping_condition <- lsc_trivial()
```

manhattan_distance *Manhattan distance*

Description

Manhattan distance

Usage

```
manhattan_distance(x, y)
```

Arguments

x	• numeric
y	• numeric

Value

numeric - manhattan distance between x and y

Examples

```
manhattan_distance(c(1, 1), c(1, 2))
```

MetaepochSnapshot-class *A S4 class representing a snapshot of one metaepoch.*

Description

A S4 class representing a snapshot of one metaepoch.

Slots

`demes` list of objects of class Deme.

`best_fitness` numeric - best fitness value of a metaepoch.

`best_solution` numeric - best solution of a metaepoch.

`time_in_seconds` numeric - time of metaepoch in seconds.

`fitness_evaluations` numeric - number of fitness evaluations.

`blocked_sprouts` list - list of sprouts that were blocked by sprouting condition. A sprout is a potential origin of a new Deme, it can be blocked by sc – sprouting condition. See `sc_max_metric` for more details.

`is_evolutionary` logical - `TRUE` for all metaepochs except the gradient one.

plot,hms-method — *Plot method for "hms" class.*

Description

Plot method for "hms" class.

Usage

```
## S4 method for signature 'hms'
plot(x)
```

Arguments

`x`	• hms s4 object

Value

It doesn't return anything meaningful. It plots the fitness by metaepoch count.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
plot(result)
```

plotActiveDemes — *plotActiveDemes method for "hms" class.*

Description

plotActiveDemes method for "hms" class.

Usage

```
plotActiveDemes(object)
```

Arguments

`object`	• hms s4 object

Value

It doesn't return anything meaningful. It plots the number of active demes per metaepoch.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
plotActiveDemes(result)
```

`plotActiveDemes,hms-method` *plotActiveDemes method for "hms" class.*

Description

plotActiveDemes method for "hms" class.

Usage

```
## S4 method for signature 'hms'
plotActiveDemes(object)
```

Arguments

`object`	• hms s4 object

Value

It doesn't return anything meaningful. It plots the number of active demes per metaepoch.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
plotActiveDemes(result)
```

`plotPopulation` *plotPopulation method for "hms" class.*

Description

plotPopulation method for "hms" class.

Usage

```
plotPopulation(object, dimensions)
```

Arguments

`object`	• hms s4 object
`dimensions`	• two selected dimensions

Value

It doesn't return anything meaningful. It plots the selected two dimensions of a population.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
plotPopulation(result, c(1, 1))
```

---

plotPopulation,hms-method

plotPopulation method for "hms" class.

Description

plotPopulation method for "hms" class.

Usage

```
## S4 method for signature 'hms'
plotPopulation(object, dimensions)
```

Arguments

`object`	• hms s4 object
`dimensions`	• two selected dimensions

Value

It doesn't return anything meaningful. It plots the selected two dimensions of a population.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
plotPopulation(result, c(1, 1))
```

print,hms-method — *Print method for class "hms".*

Description

Print method for class "hms".

Usage

```
## S4 method for signature 'hms'
print(x, ...)
```

Arguments

`x`	• hms s4 object
`...`	• other print arguments

Value

It does not return anything. The obvious side effect is output to the terminal.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
print(result)
```

printBlockedSprouts — *printBlockedSprouts method for "hms" class.*

Description

printBlockedSprouts method for "hms" class.

Usage

```
printBlockedSprouts(object)
```

Arguments

`object`	• hms s4 object

Value

It doesn’t return anything. It prints blocked sprouts per metaepoch.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
printBlockedSprouts(result)
```

`printBlockedSprouts,hms-method` *printBlockedSprouts method for "hms" class.*

Description

printBlockedSprouts method for "hms" class.

Usage

```
## S4 method for signature 'hms'
printBlockedSprouts(object)
```

Arguments

`object`	• hms s4 object

Value

It doesn’t return anything. It prints blocked sprouts per metaepoch.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
printBlockedSprouts(result)
```

`printTree` *printTree method for class "hms".*

Description

printTree method for class "hms".

Usage

```
printTree(object)
```

Arguments

`object`	• hms s4 object

Value

It does not return anything. It prints the hms tree.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
printTree(result)
```

`printTree,hms-method` *printTree method for class "hms".*

Description

printTree method for class "hms".

Usage

```
## S4 method for signature 'hms'
printTree(object)
```

Arguments

`object`	• hms s4 object

Value

It does not return anything. It prints the hms tree.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
printTree(result)
```

`rtnorm_mutation` *Factory function that creates normal mutation function*

Description

Given the domain bounds and standard deviation returns a function compatible with GA interface that performs a mutation on the given individual using truncated normal distribution.

Usage

```
rtnorm_mutation(lower, upper, sd)
```

Arguments

`lower`	• Lower bound of the problem's domain
`upper`	• Upper bound of the problem's domain
`sd`	• Standard deviation of the truncated normal distribution used for the mutation

Value

Function that takes two parameters (the GA object `object` and an individual to perform the mutation on `parent`) and returns a new individual that is the result of normal mutation applied to the parent.

Examples

```
mutation <- rtnorm_mutation(
  lower = rep(-500, 5),
  upper = rep(500, 5),
  sd = rep(50, 5)
)
```

`saveMetaepochsPopulations` *saveMetaepochsPopulations method for "hms" class.*

Description

saveMetaepochsPopulations method for "hms" class.

Usage

```
saveMetaepochsPopulations(object, path, dimensions)
```

Arguments

`object`	hms s4 object
`path`	path
`dimensions`	vector of two selected dimensions e.g. c(1,2)

Value

It doesn't return anything. It creates plots and saves them to a specified directory.

Examples

```
fitness <- function(x) x[1] + x[2]
lower <- c(-5, -5)
upper <- c(5, 5)
result <- hms(fitness = fitness, lower = lower, upper = upper)
selected_dimensions <- c(1, 2)
saveMetaepochsPopulations(result, tempdir(), selected_dimensions)
```

`saveMetaepochsPopulations,hms-method` *saveMetaepochsPopulations*

Description

saveMetaepochsPopulations

Usage

```
## S4 method for signature 'hms'
saveMetaepochsPopulations(object, path, dimensions)
```

Arguments

`object`	hms s4 object
`path`	path
`dimensions`	vector of two selected dimensions e.g. c(1,2)

Value

It doesn't return anything. It creates plots and saves them to a specified directory.

Examples

```
fitness <- function(x) x[1] + x[2]
lower <- c(-5, -5)
upper <- c(5, 5)
result <- hms(fitness = fitness, lower = lower, upper = upper)
selected_dimensions <- c(1, 2)
saveMetaepochsPopulations(result, tempdir(), selected_dimensions)
```

`sc_max_metric` *Default sprouting condition based on given metric.*

Description

It allows an individual to sprout only if there are no other demes on the target level that have centroid within the given distance.

Usage

```
sc_max_metric(metric, max_distances)
```

Arguments

`metric`	• Metric used for deme distance comparison (e.g. euclidean_distance, manhattan_distance)
`max_distances`	• numeric - maximum distance to a centroid of a deme on the target level that would allow the individual to sprout

Value

Function that can be used as a sprouting condition of hms.

Examples

```
sprouting_condition <- sc_max_metric(euclidean_distance, c(20, 10))
```

`show,hms-method` *Show method for class "hms".*

Description

Show method for class "hms".

Usage

```
## S4 method for signature 'hms'
show(object)
```

Arguments

`object`	• hms s4 object

Value

It returns the names of the slots and the classes associated with the slots in the "hms" class. It prints call details.

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
show(result)
```

---

summary,hms-method *Summary method for class "hms".*

Description

Summary method for class "hms".

Usage

```
## S4 method for signature 'hms'
summary(object, ...)
```

Arguments

object	• hms s4 object
...	• other summary arguments

Value

Returns a list with fields: fitness, solution, metaepochs, deme_population_sizes, lower_bound, upper_bound, computation_time. These fields should match fields of class "hms".

Examples

```
f <- function(x) x
result <- hms(fitness = f, lower = -5, upper = 5)
summary(result)
```

Index