Package 'rmoo'

May 4, 2026

Title Multi-Objective Optimization in R

Version 0.3.2

Date 2026-05-03

Description The 'rmoo' package is a framework for multi- and many-objective optimization, which allows researchers and users versatility in parameter configuration, as well as tools for analysis, replication and visualization of results. The 'rmoo' package was built as a fork of the 'GA' package by Luca Scrucca(2017) <DOI:10.32614/RJ-2017-008> and implementing the Non-Dominated Sorting Genetic Algorithms proposed by K. Deb's.

License GPL (>= 2)

Encoding UTF-8

Language en

LazyData true

RoxygenNote 7.3.3

Collate 'AllClasses.R' 'associate.R' 'crowding_distance.R' 'data.R' 'generate_reference_points.R' 'geneticoperator.R' 'get_fixed_rowsum_integer_matrix.R' 'miscfun.R' 'AllGenerics.R' 'niching.R' 'non_dominated_fronts.R' 'parallel.R' 'utils.R' 'nsga.R' 'nsga2.R' 'nsga3.R' 'rnsga2.R' 'modified_crowding_distance.R' 'rmooControl.R' 'performance_metrics.R' 'reference_point_multi_layer.R' 'rmoo.R' 'rmoo_main.R' 'sharing.R' 'update_points.R' 'zzz.R'

Imports stats, utils, graphics, methods, foreach, GA, grDevices, plotly, ggplot2, BBmisc

URL https://github.com/Evolutionary-Optimization-Laboratory/rmoo/

BugReports https://github.com/Evolutionary-Optimization-Laboratory/rmoo/issues/

Suggests testthat, covr, rgl, ecr, emoa, cdata, dplyr, reshape2, parallel, doParallel, doRNG (>= 1.6)

Depends R (>= 2.10)

NeedsCompilation no

Author Francisco Benitez [aut, cre],
Diego P. Pinto-Roa [aut] (ORCID:
<https://orcid.org/0000-0003-2479-9876>)

Maintainer Francisco Benitez <benitezfj94@gmail.com>

Repository CRAN

Date/Publication 2026-05-04 00:00:02 UTC

Contents

`algorithm-class` *Virtual Parent Class Algorithm*

Description

It will use when other algorithms are implemented. Equivalent to a Abstract class in other languages.

`associate` *Association Operation in Non-Dominated Genetic Algorithms III*

Description

Function that associates each member of the population with a reference point. The function calculates the perpendicular distance of each individual from each of the reference lines. This code section corresponds to Algorithm 3 of the referenced paper.

Usage

```
associate_to_niches(object, utopian_epsilon = 0)
compute_perpendicular_distance(x, y)
compute_niche_count(n_niches, niche_of_individuals)
```

Arguments

`object`	An object of class `"nsga3"`.
`utopian_epsilon`	The epsilon used for decrease the ideal point to get the utopian point.
`x`	Individuals to calculate their niche.
`y`	Reference points.
`n_niches`	Number of reference points.
`niche_of_individuals`	The niche count of individuals, except the last front.

Value

Returns a list with the niche count of individuals and the distances between them.

Author(s)

Francisco Benitez

References

J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," in IEEE Access, vol. 8, pp. 89497-89509, 2020, doi: 10.1109/ACCESS.2020.2990567.

K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

`calc_norm_pref_distance`

Calculate Normalized Preference Distance Computes the weighted normalized Euclidean distance between a set of fitness vectors and a set of reference points.

Description

Calculate Normalized Preference Distance Computes the weighted normalized Euclidean distance between a set of fitness vectors and a set of reference points.

Usage

```
calc_norm_pref_distance(fitness, ref_points, weight, ideal_point, nadir_point)
```

Arguments

`fitness`	A matrix of fitness values.
`ref_points`	A matrix of reference points.
`weight`	A numeric vector of weights for each objective.
`ideal_point`	A numeric vector of ideal point values.
`nadir_point`	A numeric vector of nadir point values.

Value

A matrix of distances where element (i, j) is the distance from fitness to ref_points.

`crowding_distance` *Calculation of Crowding Distance*

Description

A Crowded-comparison approach.

Usage

```
crowding_distance(object, nObj)
```

Arguments

`object`, `nObj`	An object of class 'nsga2', usually resulting from a call to function nsga2. Fitness Function Objective Numbers

Details

The crowded-comparison operator guides the selection process at the various stages of the algorithm toward a uniformly spread-out Pareto-optimal front

Value

A vector with the crowding-distance between individuals of a population.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II,' in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

See Also

`non_dominated_fronts()`

`generate_reference_points` *Determination of Reference Points on a Hyper-Plane*

Description

A implementation of Das and Dennis's Reference Points Generation.

Usage

```
generate_reference_points(m, h, scaling = NULL)
```

Arguments

`m`, `h`, `scaling`	Number of reference points 'h' in M-objective problems, and scaling that is the scale on which the points are distributed.

Details

The implemented Reference Point Generation is based on the Das and Dennis's systematic approach that places points on a normalized hyper-plane which is equally inclined to all objective axes and has an intercept of one on each axis.

Value

A matrix with the reference points uniformly distributed.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb and H. Jain, 'An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints,' in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

Das, Indraneel & Dennis, J. (2000). Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. SIAM Journal on Optimization. 8. 10.1137/S1052623496307510.

See Also

`non_dominated_fronts()` and `get_fixed_rowsum_integer_matrix()`

getCrowdingDistance *Accessor methods to the crowding distance for NSGA-II results*

Description

Accessor methods to the crowding distance for NSGA-II results

Usage

```
getCrowdingDistance(obj)

## S4 method for signature 'nsga2'
getCrowdingDistance(obj)
```

Arguments

obj	an object resulting from the execution of NSGA-II algorithm

Value

Returns a vector with the crowding distances of class nsga2. See nsga2 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object resulting from the execution of the NSGA-II algorithm.
#
# getCrowdingDistance(out)
#
```

getDummyFitness *Accessor methods to the dummy fitness for NSGA-I results*

Description

Accessor methods to the dummy fitness for NSGA-I results

Usage

```
getDummyFitness(obj)

## S4 method for signature 'nsga1'
getDummyFitness(obj)
```

Arguments

obj	an object resulting from the execution of NSGA-I algorithm

Value

Returns a matrix with the dummy fitness of class nsga1. See nsga1 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object resulting from the execution of the NSGA-I algorithm.
#
# getDummyFitness(out)
#
```

---

getFitness *Accessor methods to the fitness for rmoo results*

Description

Accessor methods to the fitness for rmoo results

Usage

```
getFitness(obj)
```

Arguments

obj	an object resulting from the execution of NSGA-I, NSGA-II or NSGA-III algorithm

Value

Prints the resulting fitness and when the result of the method-call is assigned to a variable, the fitness is stored as a data frame. See nsga1 nsga2, nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object resulting from the execution of the rmoo.
#
# fitness_result <- getFitness(out)
#
# fitness_result
```

`getMetrics` *Accessor methods to the metrics evaluated during execution*

Description

Accessor methods to the metrics evaluated during execution

Usage

```
getMetrics(obj)

## S4 method for signature 'nsga'
getMetrics(obj)
```

Arguments

`obj`	an object resulting from the execution of NSGA-I, NSGA-II or NSGA-III algorithm. During the execution of the performance metrics must be evaluated.

Value

A dataframe with performance metrics evaluated iteration by iteration.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object resulting from the execution of the rmoo.
#
# metrics_result <- getMetrics(out)
#
# metrics_result
```

getPopulation *Accessor methods to the population for rmoo results*

Description

Accessor methods to the population for rmoo results

Usage

```
getPopulation(obj)

## S4 method for signature 'nsga'
getPopulation(obj)

## S4 method for signature 'nsga'
getFitness(obj)
```

Arguments

obj	an object resulting from the execution of NSGA-I, NSGA-II or NSGA-III algorithm

Value

Prints the resulting population and when the result of the method-call is assigned to a variable, the population is stored as a data frame. See nsga1 nsga2, nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object resulting from the execution of rmoo.
#
# population_result <- getPopulation(out)
#
# population_result
```

`get_fixed_rowsum_integer_matrix` *Determine the division points on the hyperplane*

Description

Implementation of the recursive function in Generation of Reference points of Das and Dennis..

Usage

```
get_fixed_rowsum_integer_matrix(m, h)
```

Arguments

`m, h`	Number of reference points 'h' in M-objective problems

Details

The implemented Reference Point Generation is based on the Das and Dennis's systematic approach that places points on a normalized hyper-plane which is equally inclined to all objective axes and has an intercept of one on each axis.

Value

A matrix with the reference points uniformly distributed.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb and H. Jain, 'An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints,' in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

Das, Indraneel & Dennis, J.. (2000). Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. SIAM Journal on Optimization. 8. 10.1137/S1052623496307510.

See Also

`non_dominated_fronts()` and `generate_reference_points()`

kroA100 *KROA100*

Description

A dataset containing the coord and section of 100 cities

Usage

```
kroA100
```

Format

A data frame with 100 rows and 2 variables:

COORD City Coordinates

SECTION City Section

References

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. ORSA journal on computing, 3(4), 376-384

kroB100 *KROB100*

Description

A dataset containing the coord and section of 100 cities

Usage

```
kroB100
```

Format

A data frame with 100 rows and 2 variables:

COORD City Coordinates

SECTION City Section

References

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. ORSA journal on computing, 3(4), 376-384

kroC100 *KROC100*

Description

A dataset containing the coord and section of 100 cities

Usage

```
kroC100
```

Format

A data frame with 100 rows and 2 variables:

COORD City Coordinates

SECTION City Section

References

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. ORSA journal on computing, 3(4), 376-384

modifiedCrowdingDistance *Calculation of Modified Crowding Distance*

Description

A Crowded-comparison approach.

Usage

```
modifiedCrowdingDistance(
  object,
  epsilon,
  weights = NULL,
  normalization = "front",
  extreme_points_as_ref_dirs = FALSE
)
```

Arguments

`object`	An object of class 'rnsga2', typically from a call to r-nsga2. Must contain fitness, population, fronts, popSize, and reference_points.
`epsilon`	Minimum allowed distance between solutions to avoid duplicates.
`weights`	A numeric vector of weights for preference distance (default is equal weights).
`normalization`	Type of normalization to apply: `"ever"`, `"front"`, or `"no"`.
`extreme_points_as_ref_dirs`	Logical; whether to use extreme points as reference directions.

Details

The crowded-comparison operator maintain diversity in the Pareto front during multi-objective optimization. This version uses a reference point-based normalization and preference distance strategy.

Value

A list with:

survivors Indices of selected individuals

indexmin Index of individuals with minimum scalarizing value (optional)

reference_points Updated reference points matrix

Author(s)

Francisco Benitez

References

Kalyanmoy Deb and J. Sundar (2006). GECCO '06. doi:10.1145/1143997.1144112

See Also

`rnsga2()`

`niching` *Niche-Preservation Operation*

Description

Generation of niche, by associating reference points to population members

Usage

```
niching(pop, n_remaining, niche_count, niche_of_individuals, dist_to_niche)
```

Arguments

pop	Last Front Population
n_remaining	Number of points to choose
niche_count	Niche count of individuals with the reference point
niche_of_individuals	Count of the closest reference point to the last front objective values
dist_to_niche	Distance between closest reference point to last front objective values

Details

Niching procesure is a algorithms proposed by K. Deb and H. Jain in 2013.

Value

Returns the association of reference points to each individual in the population.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb and H. Jain, 'An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints,' in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206. doi: 10.32614/RJ-2017-008

Felix-Antoine Fortin, Francois-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagne. 2012. DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. 13, 1 (January 2012), 2171–2175.

See Also

associate_to_niches(), PerformScalarizing()

---

non_dominated_fronts *Calculate of Non-Dominated Front*

Description

A fast approach for calculate Non-Dominated Fronts.

Usage

```
non_dominated_fronts(object)
```

Arguments

object	An object of class 'nsga', usually resulting from a call to function nsga, nsga2 and nsga3.

Details

Function to determine the non-dominated fronts of a population and the aptitude value.

Value

A list with 'non-dominated fronts' and 'occupied positions' on the fronts.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II,' in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

See Also

nsga(), nsga2() and nsga3()

---

nsga *Non-Dominated Sorting in Genetic Algorithms*

Description

Minimization of a fitness function using Non-Dominated Genetic algorithms (NSGA). Local search using general-purpose optimisation algorithms can be applied stochastically to exploit interesting regions.

Usage

```
nsga(
  type = c("binary", "real-valued", "permutation"),
  fitness,
  ...,
  lower,
  upper,
  nBits,
  population = rmooControl(type)$population,
  selection = rmooControl(type)$selection,
  crossover = rmooControl(type)$crossover,
```

```
  mutation = rmooControl(type)$mutation,
  popSize = 50,
  nObj = NULL,
  dshare,
  pcrossover = 0.8,
  pmutation = 0.1,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  monitor = if (interactive()) rmooMonitor else FALSE,
  summary = FALSE,
  seed = NULL
)
```

Arguments

type	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: "binary" for binary representations of decision variables. "real-valued" for optimization problems where the decision variables are floating-point representations of real numbers. "permutation" for problems that involves reordering of a list of objects.
fitness	the fitness function, any allowable R function which takes as input an individual string representing a potential solution, and returns a numerical value describing its "fitness".
...	additional arguments to be passed to the fitness function. This allows to write fitness functions that keep some variables fixed during the search.
lower	a vector of length equal to the decision variables providing the lower bounds of the search space in case of real-valued or permutation encoded optimizations.
upper	a vector of length equal to the decision variables providing the upper bounds of the search space in case of real-valued or permutation encoded optimizations.
nBits	a value specifying the number of bits to be used in binary encoded optimizations.
population	an R function for randomly generating an initial population. See rmoo_Population() for available functions.
selection	an R function performing selection, i.e. a function which generates a new population of individuals from the current population probabilistically according to individual fitness.
crossover	an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See rmoo_Crossover() for available functions.
mutation	an R function performing mutation, i.e. a function which randomly alters the values of some genes in a parent chromosome. See rmoo_Mutation() for available functions.

popSize	the population size.
nObj	number of objective in the fitness function.
dshare	the maximun phenotypic distance allowed between any two individuals to become members of a niche.
pcrossover	the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.
pmutation	the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.
maxiter	the maximum number of iterations to run before the NSGA search is halted.
run	the number of consecutive generations without any improvement in the best fitness value before the NSGA is stopped.
maxFitness	the upper bound on the fitness function after that the NSGA search is interrupted.
names	a vector of character strings providing the names of decision variables.
suggestions	a matrix of solutions strings to be included in the initial population. If provided the number of columns must match the number of decision variables.
monitor	a logical or an R function which takes as input the current state of the nsga-class object and show the evolution of the search. By default, for interactive sessions the function rmooMonitor prints the average and best fitness values at each iteration. If set to plot these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument. In non interactive sessions, by default monitor = FALSE so any output is suppressed.
summary	If there will be a summary generation after generation.
seed	an integer value containing the random number generator state. This argument can be used to replicate the results of a NSGA search. Note that if parallel computing is required, the doRNG package must be installed.

Details

The Non-dominated genetic algorithms is a meta-heuristic proposed by N. Srinivas and K. Deb in 1994. The purpose of the algorithms is to find an efficient way to optimize multi-objectives functions (two or more).

Value

Returns an object of class nsga1-class. See nsga1 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

N. Srinivas and K. Deb, "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, in Evolutionary Computation, vol. 2, no. 3, pp. 221-248, Sept. 1994, doi: 10.1162/evco.1994.2.3.221.

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206. doi: 10.32614/RJ-2017-008

See Also

nsga2(), nsga3()

Examples

```
#Example
#Two Objectives - Real Valued
zdt1 <- function (x) {
 if (is.null(dim(x))) {
   x <- matrix(x, nrow = 1)
 }
 n <- ncol(x)
 g <- 1 + rowSums(x[, 2:n, drop = FALSE]) * 9/(n - 1)
 return(cbind(x[, 1], g * (1 - sqrt(x[, 1]/g))))
}

#Not run:
## Not run:
result <- nsga(type = "real-valued",
               fitness = zdt1,
               lower = c(0,0),
               upper = c(1,1),
               popSize = 100,
               nObj = 2,
               dshare = 1,
               monitor = FALSE,
               maxiter = 500)

## End(Not run)
```

---

nsga-class *Virtual Class 'nsga'*

Description

The 'nsga' class is the parent superclass of the nsga1, nsga2, and nsga3 classes

Slots

`call` an object of class 'call' representing the matched call.

`type` a character string specifying the type of genetic algorithm used.

`lower` a vector providing for each decision variable the lower bounds of the search space in case of real-valued or permutation encoded optimisations.

`upper` a vector providing for each decision variable the upper bounds of the search space in case of real-valued or permutation encoded optimizations.

`nBits` a value specifying the number of bits to be used in binary encoded optimizations.

`names` a vector of character strings providing the names of decision variables (optional).

`nvars` a

`popSize` the population size.

`front` Rank of individuals on the non-dominated front.

`f` Front of individuals on the non-dominated front.

`iter` the actual (or final) iteration of NSGA search.

`run` the number of consecutive generations without any improvement in the best fitness value before the NSGA is stopped.

`maxiter` the maximum number of iterations to run before the NSGA search is halted.

`suggestions` a matrix of user provided solutions and included in the initial population.

`population` the current (or final) population.

`pcrossover` the crossover probability.

`pmutation` the mutation probability.

`fitness` the values of fitness function for the current (or final) population.

`summary` a matrix of summary statistics for fitness values at each iteration (along the rows).

`fitnessValue` the best fitness value at the final iteration.

`solution` the value(s) of the decision variables giving the best fitness at the final iteration.

`execution_time` a

Objects from the Class

Since it is a virtual Class, no objects may be created from it.

Examples

```
showClass('nsga')
```

---

nsga1-class *Class 'nsga1'*

Description

The class 'nsga1' is instantiated within the execution of rmoo and will be returned as a result of it. All data generated during execution will be stored in it.

Slots

`dumFitness` a large dummy fitness value assigned to individuals from the nondominated front.

`dShare` the maximun phenotypic distance allowed between any two individuals to become members of a niche.

`deltaDummy` value to decrease the dummy fitness of individuals by non-dominated fronts.

Examples

```
showClass('nsga1')
```

nsga2 *Non-Dominated Sorting in Genetic Algorithms II*

Description

Minimization of a fitness function using non-dominated sorting genetic algorithms - II (NSGA-IIs). Multiobjective evolutionary algorithms

Usage

```
nsga2(
  type = c("binary", "real-valued", "permutation"),
  fitness,
  ...,
  lower,
  upper,
  nBits,
  population = rmooControl(type)$population,
  selection = rmooControl(type)$selection,
  crossover = rmooControl(type)$crossover,
  mutation = rmooControl(type)$mutation,
  popSize = 50,
  nObj = NULL,
  pcrossover = 0.8,
  pmutation = 0.1,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  parallel = FALSE,
  monitor = if (interactive()) rmooMonitor else FALSE,
  summary = FALSE,
  seed = NULL
)
```

Arguments

type	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: **'binary'** for binary representations of decision variables. **'real-valued'** for optimization problems where the decision variables are floating-point representations of real numbers. **'permutation'** for problems that involves reordering of a list of objects.
fitness	the fitness function, any allowable R function which takes as input an individual string representing a potential solution, and returns a numerical value describing its 'fitness'.

`...`	additional arguments to be passed to the fitness function. This allows to write fitness functions that keep some variables fixed during the search
`lower`	a vector of length equal to the decision variables providing the lower bounds of the search space in case of real-valued or permutation encoded optimizations.
`upper`	a vector of length equal to the decision variables providing the upper bounds of the search space in case of real-valued or permutation encoded optimizations.
`nBits`	a value specifying the number of bits to be used in binary encoded optimizations
`population`	an R function for randomly generating an initial population. See `rmoo_Population()` for available functions.
`selection`	an R function performing selection, i.e. a function which generates a new population of individuals from the current population probabilistically according to individual fitness.
`crossover`	an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See `rmoo_Crossover()` for available functions.
`mutation`	an R function performing mutation, i.e. a function which randomly alters the values of some genes in a parent chromosome. See `rmoo_Mutation()` for available functions.
`popSize`	the population size.
`nObj`	number of objective in the fitness function.
`pcrossover`	the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.
`pmutation`	the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.
`maxiter`	the maximum number of iterations to run before the NSGA search is halted.
`run`	the number of consecutive generations without any improvement in the best fitness value before the NSGA is stopped
`maxFitness`	the upper bound on the fitness function after that the NSGA search is interrupted.
`names`	a vector of character strings providing the names of decision variables.
`suggestions`	a matrix of solutions strings to be included in the initial population. If provided the number of columns must match the number of decision variables.
`parallel`	An optional argument which allows to specify if the NSGA-II should be run sequentially or in parallel.
`monitor`	a logical or an R function which takes as input the current state of the nsga-class object and show the evolution of the search. By default, for interactive sessions the function rmooMonitor prints the average and best fitness values at each iteration. If set to plot these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument. In non interactive sessions, by default monitor = FALSE so any output is suppressed.
`summary`	If there will be a summary generation after generation.
`seed`	an integer value containing the random number generator state. This argument can be used to replicate the results of a NSGA search. Note that if parallel computing is required, the doRNG package must be installed.

Details

The Non-dominated genetic algorithms II is a meta-heuristic proposed by K. Deb, A. Pratap, S. Agarwal and T. Meyarivan in 2002. The purpose of the algorithms is to find an efficient way to optimize multi-objectives functions (two or more).

Value

Returns an object of class nsga2-class. See nsga2 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II,' in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206. doi: 10.32614/RJ-2017-008

See Also

nsga(), nsga3()

Examples

```
#Example
#Two Objectives - Real Valued
zdt1 <- function (x) {
 if (is.null(dim(x))) {
   x <- matrix(x, nrow = 1)
 }
 n <- ncol(x)
 g <- 1 + rowSums(x[, 2:n, drop = FALSE]) * 9/(n - 1)
 return(cbind(x[, 1], g * (1 - sqrt(x[, 1]/g))))
}

#Not run:
## Not run:
result <- nsga2(type = "real-valued",
                fitness = zdt1,
                lower = c(0,0),
                upper = c(1,1),
                popSize = 100,
                nObj = 2,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

```
#Example 2
#Three Objectives - Real Valued
dtlz1 <- function (x, nobj = 3){
    if (is.null(dim(x))) {
        x <- matrix(x, 1)
    }
    n <- ncol(x)
    y <- matrix(x[, 1:(nobj - 1)], nrow(x))
    z <- matrix(x[, nobj:n], nrow(x))
    g <- 100 * (n - nobj + 1 + rowSums((z - 0.5)^2 - cos(20 * pi * (z - 0.5))))
    tmp <- t(apply(y, 1, cumprod))
    tmp <- cbind(t(apply(tmp, 1, rev)), 1)
    tmp2 <- cbind(1, t(apply(1 - y, 1, rev)))
    f <- tmp * tmp2 * 0.5 * (1 + g)
    return(f)
}

#Not run:
## Not run:
result <- nsga2(type = "real-valued",
                fitness = dtlz1,
                lower = c(0,0,0),
                upper = c(1,1,1),
                popSize = 92,
                nObj = 3,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

`nsga2-class` *Class 'nsga2'*

Description

The class 'nsga2' is instantiated within the execution of rmoo and will be returned as a result of it. All data generated during execution will be stored in it.

Slots

`crowdingDistance` Crowding-comparison approach to estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices.

Examples

```
showClass('nsga2')
```

nsga3 *Non-Dominated Sorting in Genetic Algorithms III*

Description

Minimization of a fitness function using non-dominated sorting genetic algorithms - III (NSGA-IIIs). Multiobjective evolutionary algorithms

Usage

```
nsga3(
  type = c("binary", "real-valued", "permutation"),
  fitness,
  ...,
  lower,
  upper,
  nBits,
  population = rmooControl(type)$population,
  selection = rmooControl(type)$selection,
  crossover = rmooControl(type)$crossover,
  mutation = rmooControl(type)$mutation,
  popSize = 50,
  nObj = NULL,
  n_partitions = NULL,
  pcrossover = 0.8,
  pmutation = 0.1,
  reference_dirs = generate_reference_points,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  parallel = FALSE,
  monitor = if (interactive()) rmooMonitor else FALSE,
  summary = FALSE,
  seed = NULL
)
```

Arguments

type	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: "binary" for binary representations of decision variables. "real-valued" for optimization problems where the decision variables are floating-point representations of real numbers. "permutation" for problems that involves reordering of a list of objects.

fitness	the fitness function, any allowable R function which takes as input an individual string representing a potential solution, and returns a numerical value describing its "fitness".
...	additional arguments to be passed to the fitness function. This allows to write fitness functions that keep some variables fixed during the search
lower	a vector of length equal to the decision variables providing the lower bounds of the search space in case of real-valued or permutation encoded optimizations.
upper	a vector of length equal to the decision variables providing the upper bounds of the search space in case of real-valued or permutation encoded optimizations.
nBits	a value specifying the number of bits to be used in binary encoded optimizations.
population	an R function for randomly generating an initial population. See rmoo_Population() for available functions.
selection	an R function performing selection, i.e. a function which generates a new population of individuals from the current population probabilistically according to individual fitness.
crossover	an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See rmoo_Crossover() for available functions.
mutation	an R function performing mutation, i.e. a function which randomly alters the values of some genes in a parent chromosome. See rmoo_Mutation() for available functions.
popSize	the population size.
nObj	number of objective in the fitness function.
n_partitions	Partition number of generated reference points
pcrossover	the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.
pmutation	the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.
reference_dirs	Function to generate reference points using Das and Dennis approach or matrix with supplied reference points.
maxiter	the maximum number of iterations to run before the NSGA search is halted.
run	the number of consecutive generations without any improvement in the best fitness value before the NSGA is stopped
maxFitness	the upper bound on the fitness function after that the NSGA search is interrupted.
names	a vector of character strings providing the names of decision variables.
suggestions	a matrix of solutions strings to be included in the initial population. If provided the number of columns must match the number of decision variables.
parallel	An optional argument which allows to specify if the NSGA-II should be run sequentially or in parallel.
monitor	a logical or an R function which takes as input the current state of the nsga-class object and show the evolution of the search. By default, for interactive sessions the function rmooMonitor prints the average and best fitness values at

	each iteration. If set to plot these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument. In non interactive sessions, by default monitor = FALSE so any output is suppressed.
summary	If there will be a summary generation after generation.
seed	an integer value containing the random number generator state. This argument can be used to replicate the results of a NSGA search. Note that if parallel computing is required, the doRNG package must be installed.

Details

The Non-dominated genetic algorithms III is a meta-heuristic proposed by K. Deb and H. Jain in 2013. The purpose of the algorithms is to find an efficient way to optimize multi-objectives functions (more than three).

Value

Returns an object of class nsga3-class. See nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206. doi: 10.32614/RJ-2017-008

See Also

nsga(), nsga2()

Examples

```
#Example 1
#Two Objectives - Real Valued
zdt1 <- function (x) {
 if (is.null(dim(x))) {
   x <- matrix(x, nrow = 1)
 }
 n <- ncol(x)
 g <- 1 + rowSums(x[, 2:n, drop = FALSE]) * 9/(n - 1)
 return(cbind(x[, 1], g * (1 - sqrt(x[, 1]/g))))
}

#Not run
## Not run:
```

```
result <- nsga3(type = "real-valued",
                fitness = zdt1,
                lower = c(0,0),
                upper = c(1,1),
                popSize = 100,
                nObj = 2,
                n_partitions = 100,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)

#Example 2
#Three Objectives - Real Valued
dtlz1 <- function (x, nobj = 3, ...){
    if (is.null(dim(x))) {
        x <- matrix(x, 1)
    }
    n <- ncol(x)
    y <- matrix(x[, 1:(nobj - 1)], nrow(x))
    z <- matrix(x[, nobj:n], nrow(x))
    g <- 100 * (n - nobj + 1 + rowSums((z - 0.5)^2 - cos(20 * pi * (z - 0.5))))
    tmp <- t(apply(y, 1, cumprod))
    tmp <- cbind(t(apply(tmp, 1, rev)), 1)
    tmp2 <- cbind(1, t(apply(1 - y, 1, rev)))
    f <- tmp * tmp2 * 0.5 * (1 + g)
    return(f)
}

#Not Run
## Not run:
result <- nsga3(type = "real-valued",
                fitness = dtlz1,
                lower = c(0,0,0),
                upper = c(1,1,1),
                popSize = 92,
                nObj = 3,
                n_partitions = 12,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

`nsga3-class` *Class 'nsga3'*

Description

The class 'nsga3' is instantiated within the execution of rmoo and will be returned as a result of it. All data generated during execution will be stored in it.

Slots

`ideal_point` Nadir point estimate used as lower bound in normalization.

`worst_point` Worst point generated over generations.

`smin` Index used to obtain the extreme points.

`extreme_points` are selected using the ASF in the (`PerformScalarizing()`). Necessary in the nadir point generation.

`worst_of_population` The worst individuals generated by objectives in the current generation.

`worst_of_front` The worst individuals in the first front generated by objectives in the current generation.

`nadir_point` Nadir point estimate used as upper bound in normalization.

`reference_points` NSGA-III uses a predefined set of reference points to ensure diversity in obtained solutions. The chosen refenrece points can be predefined in structured manner or supplied by the user. We use the Das and Dennis procedure.

Examples

```
showClass('nsga3')
```

---

numberOrNAOrMatrix-class

Virtual Class 'numberOrNAOrMatrix - Simple Class for subassigment Values'

Description

The class 'numberOrNAOrMatrix' is a simple class union (`setClassUnion()`) of 'numeric', 'logical', 'logical' and 'matrix'.

Objects from the Class

Since it is a virtual Class, no objects may be created from it.

Examples

```
showClass('numberOrNAOrMatrix')
```

`performance_metrics` *Objective Values performance metrics*

Description

Functions to evaluate the quality of the results obtained by the algorithms, evaluating their diversity and convergence, providing or not some parameters to compare.

Usage

```
generational_distance(front, true_pareto_front, p, inverted, plus)
```

Arguments

`front`	a N×M matrix where N is the number of points and M is the number of objectives.
`true_pareto_front`	a N×M matrix where N is the number of points and M is the number of objectives.
`p`	is the power in which the normalized distance is calculated.
`inverted`	if TRUE then computes IGD.
`plus`	if TRUE then computes the GD+.

Value

A vector with the measurement metric.

Author(s)

Francisco Benitez

References

Lamont, G., & Veldhuizen, D.V. (1999). Multiobjective evolutionary algorithms: classifications, analyses, and new innovations.

plot *Methods for Function 'plot' in Package 'rmoo'*

Description

Method used to visualize the fitness of the individuals during the execution of the algorithms.

Usage

```
plot(x, y, ...)

## S4 method for signature 'nsga,missing'
plot(x, y = "missing", type = c("scatter", "pcp", "heatmap", "polar"), ...)

## S4 method for signature 'nsga1,missing'
plot(x, y = "missing", type = c("scatter", "pcp", "heatmap", "polar"), ...)

## S4 method for signature 'nsga2,missing'
plot(x, y = "missing", type = c("scatter", "pcp", "heatmap", "polar"), ...)

## S4 method for signature 'nsga3,missing'
plot(x, y = "missing", type = c("scatter", "pcp", "heatmap", "polar"), ...)

## S4 method for signature 'rnsga2,missing'
plot(x, y = "missing", type = c("scatter", "pcp", "heatmap", "polar"), ...)
```

Arguments

x, y	Objects of either class nsga1, nsga2, or nsga3.
...	other arguments passed on to methods **"optimal"** An argument passed to the "scatter" plot. A matrix of dimension equal to the fitness with which they are compared. This value can only be compared in 2 and 3 dimensional "scatter" plots. **"individual"** An argument passed to the "heatmap" and "polar" plots. A vector that represents the fitness of the individuals to be displayed.
type	Type of graph to draw, the graphs can be of the type "scatter", "pcp", "heatmap", or "polar"

Details

The following plots are available:

- "Scatter Plot"
- "Parallel Coordinate Plot"
- "Heat Map"
- "Polar Coordinate"

Value

A graph of the evaluated type.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object of class nsga1, nsga2, or nsga3.
# The plot method will by default plot a scatter plot.
#
# plot(out)
#
# The Parallel Coordinate Plot will be plotted if "pcp" is passed as a parameter to "type".
#
# plot(out, type="pcp")
#
# A heat map plot will be plotted if "heatmap" is passed as a parameter to "type"
# and a vector with the individuals to plot to "individual"
#
# plot(out, type = "heatmap", individual = c(1:5))
#
# A polar coordinate plot will be plotted if "polar" is passed as a parameter to "type"
# and a vector with the individuals to plot to "individual"
#
# plot(out, type = "polar", individual = c(1:5))
```

---

print *Methods for Function 'print' in Package 'rmoo'.*

Description

Method used to print the slots and relevant values of the object.

Usage

```
print(x, ...)

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

## S4 method for signature 'nsga1'
print(x, ...)

## S4 method for signature 'nsga3'
print(x, ...)
```

Arguments

x	Objects of either class nsga1, nsga2, or nsga3.
...	other arguments passed on to methods

Value

Print the slots and relevant values of the object.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object of class nsga1, nsga2, or nsga3
#
# print(out)
```

---

progress *Methods for Function 'progress' in Package 'rmoo'*

Description

Method used to save the progress of the evaluation results, similar to the summary method. Passing additional arguments to the progress method evaluates performance metrics per iteration. This method cannot be called outside of rmoo execution.

Usage

```
progress(object, ...)

## S4 method for signature 'nsga'
progress(object, ...)

## S4 method for signature 'nsga1'
progress(object, ...)

## S4 method for signature 'nsga2'
progress(object, ...)

## S4 method for signature 'nsga3'
progress(object, ...)
```

Arguments

object	Objects of either class nsga1, nsga2, or nsga3.
...	other arguments passed on to methods. Passing "reference_dirs" as arguments will evaluate the performance metrics Hypervolumen, Generational Distance, and Inverse Generational Distance.

Value

A list of length equal to the number of iterations, where the progress made during execution is saved.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object of class nsga1, nsga2, or nsga3, and callArgs are
# the additional arguments passed when calling the rmoo function, for the
# evaluation of performance metrics, reference points are expected to be passed
# as an argument to reference_dirs.
#
# progress(object, callArgs)
#
```

---

reference_point_multi_layer

Determination of Multi-layer Reference Points

Description

A implementation of Multi-layer Reference Points Generation.

Usage

```
reference_point_multi_layer(...)
```

Arguments

...	The different layers provided by the user

Details

The Multi-layer reference point implementation is based on Blank and Deb's pymoo library, the approach generates different layers of references point at different scales, provided by the user.

Value

A matrix with the multi-layer reference points

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," in IEEE Access, vol. 8, pp. 89497-89509, 2020, doi: 10.1109/ACCESS.2020.2990567.

Das, Indraneel & Dennis, J. (2000). Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. SIAM Journal on Optimization. 8. 10.1137/S1052623496307510.

See Also

`generate_reference_points()` and `get_fixed_rowsum_integer_matrix()`

`rmooControl` *A function for setting or retrieving defaults non-dominated genetic operators*

Description

Default settings for non-dominated genetic operators used in the 'rmoo' package.

Usage

```
rmooControl(...)
```

Arguments

...	no arguments, a single character vector, or a named list with components.

Details

If the function is called with no arguments returns the current default settings, i.e., a list with the following default components:

`"binary"`
- `population = "rmoobin_Population"`
- `selection = "rmoobin_tourSelection"`
- `crossover = "rmoobin_spCrossover"`
- `mutation = "rmoobin_raMutation"`

`"real-valued"`
- `population = "rmooreal_Population"`
- `selection = "rmooreal_tourSelection"`
- `crossover = "rmooreal_sbxCrossover"`

- mutation = "rmooreal_polMutation"

"permutation"
- population = "rmooperm_Population"
- selection = "rmooperm_tourSelection"
- crossover = "rmooperm_oxCrossover"
- mutation = "rmooperm_simMutation"

"discrete"
- population = "rmooint_Population"
- selection = "rmooint_tourSelection"
- crossover = "rmooint_uxCrossover"
- mutation = "rmooint_uxMutation"

"eps" = the tolerance value used by the package functions. By default set at `sqrt(.Machine$double.eps)`.

The function may be called with a single string specifying the name of the component. In this case the function returns the current default settings.

To change the default values, a named component must be followed by a single value (in case of "eps") or a list of component(s) specifying the name of the function for a genetic operator. See the Examples section.

Value

If the argument list is empty the function returns the current list of values. If the argument list is not empty, the returned list is invisible.

Note

The parameter values set via a call to this function will remain in effect for the rest of the session, affecting the subsequent behaviour of the functions for which the given parameters are relevant.

Author(s)

Francisco Benitez

References

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206, doi: 10.32614/RJ-2017-008.

See Also

`nsga2()`, `rnsga2()` and `nsga3()`

Examples

```
# get and save defaults
defaultControl <- rmooControl()
print(defaultControl)
# get current defaults only for real-valued search
rmooControl("real-valued")
# set defaults for selection operator of real-valued search
rmooControl("real-valued" = list(selection = "rmooreal_lrSelection"))
```

```
rmooControl("real-valued")
# set defaults for selection and crossover operators of real-valued search
rmooControl("real-valued" = list(selection = "rmooreal_lrSelection",
                                 crossover = "rmooreal_spCrossover"))
rmooControl("real-valued")
# restore defaults
rmooControl(defaultControl)
rmooControl()
```

`rmooMonitor` *Monitor the execution of rmoo*

Description

Functions to plotting fitness values at each iteration of a search for the 'rmoo' package.

Usage

```
rmooMonitor(object, ...)
```

Arguments

`object`	an object of class `nsga`, `nsga2` or `nsga3`, usually resulting from a call to function `nsga`, `nsga2` or `nsga3`, respectively.
`...`	further arguments passed to or from other methods.

Value

These functions plot the fitness values of the current step of the nsga3 on the console.
By default, `rmooMonitor` is called in interactive sessions by `nsga`, `nsga2`, or `nsga3`.
The function can be modified by the user to plot or print the values it considers by iteration.

Author(s)

Francisco Benitez

References

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206, doi: 10.32614/RJ-2017-008.

See Also

`nsga()`, `nsga2()` and `nsga3()`

`rmooreal_sbxCrossover` *Crossover Operators in Non-Dominated Genetic Algorithms*

Description

Functions implementing crossover operators for non-dominated genetic algorithms. `rmoo_spCrossover` (and its typed variants `rmoobin_`, `rmooreal_`, `rmooint_`) performs single-point crossover; `rmooreal_sbxCrossover` performs simulated binary crossover; `rmooperm_oxCrossover` performs order crossover for permutation representations.

Usage

```
rmooreal_sbxCrossover(object, parents, eta = 20, indpb = 0.5)

rmoo_spCrossover(object, parents)

rmoobin_spCrossover(object, parents)

rmooreal_spCrossover(object, parents)

rmooint_spCrossover(object, parents)

rmooperm_oxCrossover(object, parents)
```

Arguments

`object`	An object of class `"nsga"`, `"nsga2"`, or `"nsga3"`, usually from a call to `nsga`, `nsga2`, or `nsga3`.
`parents`	A two-element integer vector indexing the parents from the current population.
`eta`	The distribution index. A higher eta produces offspring closer to the parents, while a lower eta allows for larger differences.
`indpb`	The probability of a particular gene being crossed.

Value

A list with two elements:

children A matrix of dimension 2 × nVars containing the generated offspring.

fitness A 2 × nObj matrix of `NA` values, indicating that offspring fitness has not yet been evaluated.

Author(s)

Francisco Benitez

References

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. *The R Journal*, 9/1, 187–206, doi:10.32614/RJ2017008.

See Also

nsga(), nsga2(), nsga3()

---

rmoo_huxCrossover *Half Uniform Crossover (HUX)*

Description

Produces two children by swapping exactly half of the genes that differ between the two parents. Only loci where the parents disagree are eligible, making HUX more conservative than full uniform crossover.

Usage

```
rmoo_huxCrossover(object, parents, prob_hux = 0.5)

rmooint_huxCrossover(object, parents, prob_hux = 0.5)

rmoobin_huxCrossover(object, parents, prob_hux = 0.5)
```

Arguments

object	MOEA object with slots population and fitness.
parents	Integer vector of length 2 with the parent row indices.
prob_hux	Proportion of differing loci to swap (default 0.5).

Value

A list with children (2 × n integer matrix) and fitness (2 × nObj NA matrix).

---

rmoo_lrSelection *Linear Rank Selection*

Description

Selects individuals from the population using linear rank-based probabilities. Individuals are ranked by their Pareto front, and selection probability is assigned linearly so better-ranked individuals are more likely to be chosen.

Usage

```
rmoo_lrSelection(object, r, q)

rmoobin_lrSelection(object, r, q)

rmooperm_lrSelection(object, r, q)

rmooreal_lrSelection(object, r, q)
```

Arguments

object	MOEA object with slots population, fitness, front, popSize.
r	Slope parameter controlling probability spread (computed from popSize if missing).
q	Intercept parameter (computed from popSize if missing).

Value

A list with population and fitness of the selected individuals.

---

rmoo_main *R Multi-Objective Optimization Main Function*

Description

Main function of rmoo, based on the parameters it will call the different algorithms implemented in the package. Optimization algorithms will minimize a fitness function. For more details of the algorithms see nsga2(), nsga3(), rnsga2().

Usage

```
rmoo(
  type = c("binary", "real-valued", "permutation", "discrete"),
  algorithm = c("NSGA-II", "NSGA-III", "R-NSGA-II"),
  fitness,
  ...,
  lower,
  upper,
  nBits,
  nvars,
  population = rmooControl(type)$population,
  selection = rmooControl(type)$selection,
  crossover = rmooControl(type)$crossover,
  mutation = rmooControl(type)$mutation,
  pcrossover = 0.8,
  pmutation = 0.1,
  popSize = 50,
  maxiter = 100,
  nObj = NULL,
  names = NULL,
  suggestions = NULL,
  monitor = if (interactive()) rmooMonitor else FALSE,
  parallel = FALSE,
  summary = FALSE,
  seed = NULL,
  reference_dirs = NULL,
```

```
  epsilon = 0.001,
  normalization = NULL,
  extreme_points_as_ref_dirs = FALSE,
  weights = NULL
)
```

Arguments

type	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: **'binary'** for binary representations of decision variables. **'real-valued'** for optimization problems where the decision variables are floating-point representations of real numbers. **'permutation'** for problems that involves reordering of a list of objects. **'discrete'** for discrete representations of decision variables.
algorithm	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: **'NSGA-II'** for . **'NSGA-III'** for . **'R-NSGA-II'** for .
fitness	the fitness function, any allowable R function which takes as input an individual string representing a potential solution, and returns a numerical value describing its 'fitness'.
...	argument in which all the values necessary for the configuration will be passed as parameters. The user is encouraged to see the documentations.
lower	a vector of length equal to the decision variables providing the lower bounds of the search space in case of real-valued or permutation encoded optimizations.
upper	a vector of length equal to the decision variables providing the upper bounds of the search space in case of real-valued or permutation encoded optimizations.
nBits	a value specifying the number of bits to be used in binary encoded optimizations.
nvars	a value .
population	an R function for randomly generating an initial population. See rmoo_Population() for available functions.
selection	an R function performing selection, i.e. a function which generates a new population of individuals from the current population probabilistically according to individual fitness.
crossover	an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See rmoo_Crossover() for available functions.
mutation	an R function performing mutation, i.e. a function which randomly alters the values of some genes in a parent chromosome. See rmoo_Mutation() for available functions.
pcrossover	the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.

`pmutation` the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.

`popSize` the population size.

`maxiter` the maximum number of iterations to run before the NSGA search is halted.

`nObj` number of objective in the fitness function.

`names` a vector of character strings providing the names of decision variables.

`suggestions` a matrix of solutions strings to be included in the initial population. If provided the number of columns must match the number of decision variables.

`monitor` a logical or an R function which takes as input the current state of the nsga-class object and show the evolution of the search. By default, for interactive sessions the function rmooMonitor prints the average and best fitness values at each iteration. If set to plot these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument. In non interactive sessions, by default monitor = FALSE so any output is suppressed.

`parallel` An optional argument which allows to specify if the NSGA-II should be run sequentially or in parallel.

`summary` If there will be a summary generation after generation.

`seed` an integer value containing the random number generator state. This argument can be used to replicate the results of a NSGA search. Note that if parallel computing is required, the doRNG package must be installed.

`reference_dirs` Function to generate reference points using Das and Dennis approach or matrix with supplied reference points.

`epsilon` controls the extent of obtained solutions by grouping all solutions that have a normalized difference sum in objective values of epsilon or less.

`normalization` of the ideal points and nadir. They can be:

'ever' .

'front' .

'no' .

`extreme_points_as_ref_dirs`

flag to use extreme points as reference points.

`weights` vector specifies the importance of one objective function over the other, by default all objectives have equal weights. of `nsga2()`, `rnsga2()`, `nsga3()` in which the necessary parameters for each algorithm are cited, in addition, the chosen strategy to execute must be passed as an argument. This can be seen more clearly in the examples.

Details

Multi- and Many-Optimization of a fitness function using Non-dominated Sorting Genetic Algorithms. The algorithms currently implemented by rmoo are: NSGA-II, NSGA-III and R-NSGA-II

The Non-dominated genetic algorithms II (NSGA-II) is a meta-heuristic proposed by K. Deb, A. Pratap, S. Agarwal and T. Meyarivan in 2002. The purpose of the algorithms is to find an efficient way to optimize multi-objectives functions (two or more).

The Non-dominated genetic algorithms III (NSGA-III) is a meta-heuristic proposed by K. Deb and H. Jain in 2013. The purpose of the algorithms is to find an efficient way to optimize multi-objectives functions (more than three).

The Reference point-based Non-dominated genetic algorithms II (R-NSGA-II) is a meta-heuristic proposed by K. Deb and J. Sundar in 2006. It is a modification of NSGA-II based on reference points in which the decision-maker supplies one or more preference points and a weight vector that will guide the solutions towards regions desired by the user.

Value

Returns an object of class nsga2-class, rnsga2-class or nsga3-class. See nsga2, rnsga2, nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206. doi: 10.32614/RJ-2017-008

Kalyanmoy Deb and J. Sundar. 2006. Reference point based multi-objective optimization using evolutionary algorithms. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06). Association for Computing Machinery, New York, NY, USA, 635–642. doi: 10.1145/1143997.1144112

K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II,' in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

See Also

nsga2(), rnsga2(), nsga3()

Examples

```
#Example 1
#Two Objectives - Real Valued
zdt1 <- function (x) {
 if (is.null(dim(x))) {
   x <- matrix(x, nrow = 1)
 }
 n <- ncol(x)
 g <- 1 + rowSums(x[, 2:n, drop = FALSE]) * 9/(n - 1)
 return(cbind(x[, 1], g * (1 - sqrt(x[, 1]/g))))
}
```

```
#Not run:
## Not run:
result <- rmoo(type = "real-valued",
               fitness = zdt1,
               algorithm = "NSGA-II",
               lower = c(0,0),
               upper = c(1,1),
               popSize = 100,
               nObj = 2,
               monitor = FALSE,
               maxiter = 500)


## End(Not run)

#Example 2
#Three Objectives - Real Valued
dtlz1 <- function (x, nobj = 3){
    if (is.null(dim(x))) {
        x <- matrix(x, 1)
    }
    n <- ncol(x)
    y <- matrix(x[, 1:(nobj - 1)], nrow(x))
    z <- matrix(x[, nobj:n], nrow(x))
    g <- 100 * (n - nobj + 1 + rowSums((z - 0.5)^2 - cos(20 * pi * (z - 0.5))))
    tmp <- t(apply(y, 1, cumprod))
    tmp <- cbind(t(apply(tmp, 1, rev)), 1)
    tmp2 <- cbind(1, t(apply(1 - y, 1, rev)))
    f <- tmp * tmp2 * 0.5 * (1 + g)
    return(f)
}

#Define uniformly distributed reference points.
ref_points <- generate_reference_points(3,12)

#Not Run
## Not run:
result <- rmoo(type = "real-valued",
                fitness = dtlz1,
                algorithm = "NSGA-III",
                lower = c(0,0,0),
                upper = c(1,1,1),
                popSize = 92,
                nObj = 3,
                reference_dirs = ref_points,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)

#Example 3
#Two Objectives - Real Valued with Preference-guided
```

```
zdt2 <- function (x)
{
  if (is.null(dim(x))) {
    x <- matrix(x, nrow = 1)
  }
  n <- ncol(x)
  g <- 1 + rowSums(x[, 2:n, drop = FALSE]) * 9/(n - 1)
  return(cbind(x[, 1], g * (1 - (x[, 1]/g)^2)))
}

#Define uniformly distributed reference points.
ref_points <- rbind(c(1.0, 0.0), c(0.0, 1.0), c(0.5, 0.5))

#Not run
## Not run:
result <- rmoo(type = "real-valued",
               fitness = zdt2,
               algorithm = "R-NSGA-II",
               lower = c(0,0),
               upper = c(1,1),
               reference_dirs = ref_points,
               popSize = 92,
               nObj = 2,
               monitor = FALSE,
               maxiter = 500)


## End(Not run)
```

`rmoo_Mutation` *Mutation operators in non-dominated genetic algorithms*

Description

Functions implementing mutation non-dominated genetic operator.

Usage

```
rmoobin_raMutation(object, parent)

rmooreal_raMutation(object, parent)
rmooreal_polMutation(object, parent, eta = 20, indpb = 0.5)

rmooperm_simMutation(object, parent)
```

Arguments

`object`	An object of class `"nsga"`, `"nsga2"` or `"nsga3"` usually resulting from a call to function `nsga`, `nsga2`, `nsga3`.

parent	A vector of values for the parent from the current population where mutation should occur.
eta	Distribution parameter for mutation operator.
indpb	Independent mutation probability.

Value

Return a vector of values containing the mutated string.

Author(s)

Francisco Benitez

References

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206, doi: 10.32614/RJ-2017-008.

---

rmoo_Population *Population initialization in non-dominated genetic algorithms*

Description

Functions for creating a random initial population to be used in non-dominated genetic algorithms.

Usage

```
rmoobin_Population(object)

rmooreal_Population(object)

rmooperm_Population(object)

rmooint_Population(object)
```

Arguments

object	An object of class nsga-class, nsga2-class or nsga3-class.

Details

rmoobin_Population generates a random population of object@nBits binary values;

rmooreal_Population generates a random (uniform) population of real values in the range [object@lower, object@upper];

rmooperm_Population generates a random (uniform) population of permutation values in the range [object@lower, object@upper].

rmooint_Population generates a random (uniform) population of integer values in the range [object@lower, object@upper].

Value

Return a matrix of dimension `object@popSize` times the number of decision variables.

Author(s)

Francisco Benitez

References

Scrucca, L. (2017) On some extensions to 'GA' package: hybrid optimisation, parallelisation and islands evolution. The R Journal, 9/1, 187-206, doi: 10.32614/RJ-2017-008.

See Also

`nsga`, `nsga2` and `nsga3`

`rmoo_tourSelection` *Tournament Selection*

Description

Binarily o por pref

Usage

```
rmoo_tourSelection(object, k = 2, ...)

rmooreal_tourSelection(object, k = 2, ...)

rmoobin_tourSelection(object, k = 2, ...)

rmooperm_tourSelection(object, k = 2, ...)
```

Arguments

`object`	MOEA object with slots population, fitness, front, popSize.
`k`	Tournament size.
`...`	Argument which all the values necessary for the configuration will be passed as parameters. The user is encouraged to see the documentations.

Value

List with `population` and `fitness` subsets.

Examples

```
## Not run:
# Creamos un "dummy" objeto mínimo
object <- list(
  population       = matrix(runif(20), nrow=5),
  fitness          = matrix(runif(10), nrow=5),
  front            = matrix(sample(1:2, 5, TRUE), ncol=1),
  crowdingDistance = runif(5),
  popSize          = 5
)
class(object) <- "nsga2"
# Llamamos al selector
sel <- rmoo_tourSelection(object, k = 2)
str(sel)

## End(Not run)
```

`rmoo_uxCrossover` *Uniform Crossover*

Description

Produces two children by randomly swapping genes between two parents with equal probability at each locus. Each gene is inherited from either parent independently, giving maximum gene-level mixing.

Usage

```
rmoo_uxCrossover(object, parents)

rmooint_uxCrossover(object, parents)

rmoobin_uxCrossover(object, parents)
```

Arguments

`object`	MOEA object with slots population and fitness.
`parents`	Integer vector of length 2 with the parent row indices.

Value

A list with `children` (2 × n integer matrix) and `fitness` (2 × nObj NA matrix).

rmoo_uxMutation *Uniform Mutation*

Description

Mutates an individual by randomly replacing each gene with a new value drawn uniformly from the integer range [lower, upper], independently at each locus with probability indpb.

Usage

```
rmoo_uxMutation(object, parent, indpb = 0.1)

rmooint_uxMutation(object, parent, indpb = 0.1)

rmoobin_uxMutation(object, parent, indpb = 0.1)
```

Arguments

object	MOEA object with slots population, lower, upper.
parent	Index of the individual to mutate.
indpb	Per-gene mutation probability (default 0.1).

Value

An integer vector of the mutated individual.

rnsga2 *Reference Point Based Non-Dominated Sorting in Genetic Algorithms II*

Description

Minimization of a fitness function using reference point based non-dominated sorting genetic algorithms - II (R-NSGA-IIs). Multiobjective evolutionary algorithms

Usage

```
rnsga2(
  type = c("binary", "real-valued", "permutation"),
  fitness,
  ...,
  lower,
  upper,
  nBits,
  population = rmooControl(type)$population,
```

```
  selection = rmooControl(type)$selection,
  crossover = rmooControl(type)$crossover,
  mutation = rmooControl(type)$mutation,
  reference_dirs = NULL,
  epsilon = 0.001,
  normalization = c("ever", "front", "no"),
  extreme_points_as_ref_dirs = FALSE,
  weights = NULL,
  popSize = 50,
  nObj = NULL,
  pcrossover = 0.8,
  pmutation = 0.1,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  parallel = FALSE,
  monitor = if (interactive()) rmooMonitor else FALSE,
  summary = FALSE,
  seed = NULL
)
```

Arguments

type	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: **'binary'** for binary representations of decision variables. **'real-valued'** for optimization problems where the decision variables are floating-point representations of real numbers. **'permutation'** for problems that involves reordering of a list of objects.
fitness	the fitness function, any allowable R function which takes as input an individual string representing a potential solution, and returns a numerical value describing its 'fitness'.
...	additional arguments to be passed to the fitness function. This allows to write fitness functions that keep some variables fixed during the search
lower	a vector of length equal to the decision variables providing the lower bounds of the search space in case of real-valued or permutation encoded optimizations.
upper	a vector of length equal to the decision variables providing the upper bounds of the search space in case of real-valued or permutation encoded optimizations.
nBits	a value specifying the number of bits to be used in binary encoded optimizations
population	an R function for randomly generating an initial population. See rmoo_Population() for available functions.
selection	an R function performing selection, i.e. a function which generates a new population of individuals from the current population probabilistically according to individual fitness.

`crossover` an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See `rmoo_Crossover()` for available functions.

`mutation` an R function performing mutation, i.e. a function which randomly alters the values of some genes in a parent chromosome. See `rmoo_Mutation()` for available functions.

`reference_dirs` Function to generate reference points using Das and Dennis approach or matrix with supplied reference points.

`epsilon` controls the extent of obtained solutions by grouping all solutions that have a normalized difference sum in objective values of epsilon or less.

`normalization` of the ideal points and nadir. They can be:

'ever' .

'front' .

'no' .

`extreme_points_as_ref_dirs`
flag to use extreme points as reference points.

`weights` vector specifies the importance of one objective function over the other, by default all objectives have equal weights.

`popSize` the population size.

`nObj` number of objective in the fitness function.

`pcrossover` the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.

`pmutation` the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.

`maxiter` the maximum number of iterations to run before the NSGA search is halted.

`run` the number of consecutive generations without any improvement in the best fitness value before the NSGA is stopped

`maxFitness` the upper bound on the fitness function after that the NSGA search is interrupted.

`names` a vector of character strings providing the names of decision variables.

`suggestions` a matrix of solutions strings to be included in the initial population. If provided the number of columns must match the number of decision variables.

`parallel` An optional argument which allows to specify if the NSGA-II should be run sequentially or in parallel.

`monitor` a logical or an R function which takes as input the current state of the nsga-class object and show the evolution of the search. By default, for interactive sessions the function rmooMonitor prints the average and best fitness values at each iteration. If set to plot these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument. In non interactive sessions, by default monitor = FALSE so any output is suppressed.

`summary` If there will be a summary generation after generation.

`seed` an integer value containing the random number generator state. This argument can be used to replicate the results of a NSGA search. Note that if parallel computing is required, the doRNG package must be installed.

Details

R-NSGA-II is a meta-heuristic proposed by K. Deb and J. Sundar in 2006. It is a modification of NSGA-II based on reference points in which the decision-maker supplies one or more preference points and a weight vector that will guide the solutions towards regions desired by the user.

Value

Returns an object of class rnsga2-class. See rnsga2 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

Kalyanmoy Deb and J. Sundar. 2006. Reference point based multi-objective optimization using evolutionary algorithms. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06). Association for Computing Machinery, New York, NY, USA, 635–642. doi: 10.1145/1143997.1144112

See Also

nsga(), nsga2(), nsga3()

Examples

```
#Example
#Two Objectives - Real Valued
zdt1 <- function (x) {
 if (is.null(dim(x))) {
   x <- matrix(x, nrow = 1)
 }
 n <- ncol(x)
 g <- 1 + rowSums(x[, 2:n, drop = FALSE]) * 9/(n - 1)
 return(cbind(x[, 1], g * (1 - sqrt(x[, 1]/g))))
}

#Define the reference points
reference_points = rbind(c(0.2, 0.8), c(0.8, 0.2), c(0.4, 0.5))

#Not run:
## Not run:
result <- rnsga2(type = "real-valued",
                 fitness = zdt1,
                 lower = c(0,0),
                 upper = c(1,1),
                 reference_dirs = reference_points,
                 popSize = 100,
                 nObj = 2,
                 monitor = FALSE,
                 maxiter = 500,
```

```
                seed = 45)

## End(Not run)

#Example 2
#Three Objectives - Real Valued
dtlz1 <- function (x, nobj = 3){
    if (is.null(dim(x))) {
        x <- matrix(x, 1)
    }
    n <- ncol(x)
    y <- matrix(x[, 1:(nobj - 1)], nrow(x))
    z <- matrix(x[, nobj:n], nrow(x))
    g <- 100 * (n - nobj + 1 + rowSums((z - 0.5)^2 - cos(20 * pi * (z - 0.5))))
    tmp <- t(apply(y, 1, cumprod))
    tmp <- cbind(t(apply(tmp, 1, rev)), 1)
    tmp2 <- cbind(1, t(apply(1 - y, 1, rev)))
    f <- tmp * tmp2 * 0.5 * (1 + g)
    return(f)
}

#Define the reference points
reference_points <- rbind(c(1.0, 0.5, 0.0), c(0.0, 0.5, 1.0), c(0.5, 0.5, 0.5))

#Not run:
## Not run:
result <- rnsga2(type = "real-valued",
                fitness = dtlz1,
                lower = c(0,0,0),
                upper = c(1,1,1),
                reference_dirs = reference_points,
                popSize = 92,
                nObj = 3,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

`rnsga2-class` *Class 'rnsga2'*

Description

The class 'rnsga2' is instantiated within the execution of rmoo and will be returned as a result of it. All data generated during execution will be stored in it.

Slots

`crowdingDistance` Crowding-comparison approach to estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices.

`reference_points` R-NSGA-II uses a set of reference points defined by the user to ensure diversity in obtained solutions.

`extreme_points` are selected using the ASF in the (`PerformScalarizing()`). Necessary in the nadir point generation.

`smin` Index used to obtain the extreme points.

Examples

```
showClass('rnsga2')
```

`scale_reference_directions` *Scale Reference Points*

Description

A implementation of Das and Dennis's Reference Points Generation.

Usage

```
scale_reference_directions(ref_dirs, scaling)
```

Arguments

`ref_dirs, scaling` where 'ref_dirs' are the reference points generated and 'scaling' are the scale on which the points are distributed.

Details

The implemented Reference Point Generation is based on the Das and Dennis's systematic approach that places points on a normalized hyper-plane which is equally inclined to all objective axes and has an intercept of one on each axis.

Value

A matrix with rescaled reference points uniformly distributed.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," in IEEE Access, vol. 8, pp. 89497-89509, 2020, doi: 10.1109/ACCESS.2020.2990567.

See Also

generate_reference_points() and get_fixed_rowsum_integer_matrix()

---

sharing *Calculation of Dummy Fitness*

Description

Calculate of sharing distance and dummy fitness

Usage

```
sharing(object)
```

Arguments

object	An object of class 'nsga', usually resulting from a call to function nsga. Fitness Function Objective Numbers.

Details

The sharing distance operator guides the selection process at the various stages of the algorithm toward a uniformly spread-out Pareto-optimal front

Value

A vector with the dummy fitness.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

References

N. Srinivas and K. Deb, 'Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms,' in Evolutionary Computation, vol. 2, no. 3, pp. 221-248, Sept. 1994, doi: 10.1162/evco.1994.2.3.221.

See Also

non_dominated_fronts()

startParallel *Start Parallel Backend for rmoo Package*

Description

This function sets up parallel computing using the `parallel` and `doParallel` packages. It supports both "snow" (PSOCK) and "multicore" backends depending on the OS.

Usage

```
startParallel(parallel = TRUE, ...)
```

Arguments

`parallel`	Logical, numeric, character, or a cluster object. If `TRUE`, uses all detected cores. If numeric, it specifies the number of cores. If character, should be `"snow"` or `"multicore"`. If a cluster object, it will use that cluster.
`...`	Additional arguments (currently unused).

Value

An object of class `logical` with attributes:

- `type`: cluster type ("snow" or "multicore")
- `cores`: number of cores used
- `cluster`: the cluster object created or passed

Examples

```
## Not run:
  cl <- startParallel(TRUE)
  stopParallel(attr(cl, "cluster"))

## End(Not run)
```

stopParallel *Stop Parallel Backend*

Description

Stops the parallel backend and reverts to sequential execution.

Usage

```
stopParallel(cluster, ...)
```

Arguments

`cluster`	A cluster object, typically retrieved from `attr(parallel, "cluster")`
`...`	Additional arguments (currently unused).

Value

`invisible(NULL)`, used for side effects.

Examples

```
## Not run:
  cl <- startParallel()
  stopParallel(attr(cl, "cluster"))

## End(Not run)
```

---

summary *Methods for Function 'summary' in Package 'rmoo'*

Description

Method used to summarize the results of the evaluations, passing additional arguments in the summary method the performance metrics is evaluated.

Usage

```
summary(object, ...)

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

## S4 method for signature 'nsga1'
summary(object, ...)

## S4 method for signature 'nsga2'
summary(object, ...)

## S4 method for signature 'nsga3'
summary(object, ...)
```

Arguments

`object`	Objects of either class nsga1, nsga2, or nsga3.
`...`	other arguments passed on to methods. Passing `"reference_dirs"` as arguments will evaluate the performance metrics Hypervolumen, Generational Distance, and Inverse Generational Distance.

Value

A summary of the values resulting from the execution of an algorithm.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
# Where 'out' is an object of class nsga1, nsga2, or nsga3
#
# summary(out)
#
# For the evaluation of the metrics, pass the reference point
#
# ref_points <- generate_reference_points(3,12)
# summary(out, reference_dirs = ref_points)
```

---

update_points *Adaptive normalization of population members*

Description

Functions to scalarize the members of the population to locate them in a normalized hyperplane, finding the ideal point, nadir point, worst point and the extreme points.

Usage

```
UpdateIdealPoint(object, nObj)
UpdateWorstPoint(object, nObj)
PerformScalarizing(population, fitness, smin, extreme_points, ideal_point)
get_nadir_point(object)
```

Arguments

object	An object of class "nsga3".
nObj	numbers of objective values of the function to evaluate.
population	individuals of the population until last front.
fitness	objective values of the population until last front.
smin	Achievement Escalation Function Index.
extreme_points	Extreme points of the previous generation to upgrade.
ideal_point	Ideal point of the current generation to translate objectives.

Value

Return scalarized objective values in a normalized hyperplane.

Author(s)

Francisco Benitez

References

J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," in IEEE Access, vol. 8, pp. 89497-89509, 2020, doi: 10.1109/ACCESS.2020.2990567.

K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, Aug. 2014, doi: 10.1109/TEVC.2013.2281535.

Index