Package 'rmoo'

September 7, 2021

Title Multi-Objective Optimization in R

Version 0.1.7

Description A multiobjective optimization package based on K. Deb's algorithm and inspired in 'GA' package by Luca Scrucca (2017) <DOI:10.32614/RJ-2017-008>. The 'rmoo' package is a framework for multi- and many-objective optimization, allowing to work with representation of real numbers, permutations and binaries, offering a high range of configurations.

License GPL (>= 2)

Encoding UTF-8

Language es

LazyData true

RoxygenNote 7.1.1

Collate 'associate.R' 'crowding_distance.R' 'data.R' 'generate_reference_points.R' 'geneticoperator.R' 'get_fixed_rowsum_integer_matrix.R' 'miscfun.R' 'niching.R' 'non_dominated_fronts.R' 'numberOrNAOrMatrix-class.R' 'nsga-class.R' 'nsga.R' 'nsga2-class.R' 'nsga2.R' 'nsga3-class.R' 'nsga3.R' 'nsgaControl.R' 'reference_point_multi_layer.R' 'rmoo.R' 'performance_metrics.R' 'plotting.R' 'sharing.R' 'update_points.R' 'zzz.R'

Imports stats, utils, ecr, graphics, methods, grDevices, ggplot2, reshape2, dplyr, cdata, plotly

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

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

Suggests testthat, covr, rgl

Depends R (>= 2.10)

NeedsCompilation no

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

Maintainer Francisco Benitez <benitezfj94@gmail.com>

Repository CRAN

Date/Publication 2021-09-07 04:30:02 UTC

R topics documented:

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

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.

`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()`

`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()`

heat_map *Heatmap Plots*

Description

The heat_map() function for hyperdimensional data visualization, which shows magnitude of a phenomenon as color in two dimension.

Usage

```
heat_map(fitness)
```

Arguments

fitness	An matrix of values representing the fitness of the objective values of nsga-class, nsga2-class or nsga3-class. See nsga, nsga2 or nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
#Four Objectives Plotting
dtlz1 <- function (x, nobj = 4){
    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 = rep(0,4),
                upper = rep(1,4),
                popSize = 92,
                n_partitions = 12,
                monitor = FALSE,
```

```
                maxiter = 500)

## End(Not run)
#Not Run
## Not run:
heat_map(fitness = result@fitness)

## End(Not run)
```

`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

`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 = nsgaControl(type)$population,
  selection = nsgaControl(type)$selection,
  crossover = nsgaControl(type)$crossover,
  mutation = nsgaControl(type)$mutation,
  popSize = 50,
  nObj = ncol(fitness(matrix(10000, ncol = 100, nrow = 100))),
  dshare,
  pcrossover = 0.8,
  pmutation = 0.1,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  monitor = if (interactive()) nsgaMonitor 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 nsga_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. See nsga_Selection() for available functions.
crossover	an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See nsga_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 nsga_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 nsgaMonitor 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 nsga-class. See nsga 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,
               dshare = 1,
```

```
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

`nsga-class` *Virtual Class 'nsga - Simple Class for subassigment Values'*

Description

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

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).

`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.

`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.

`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.

Objects from the Class

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

Examples

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

---

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 = nsgaControl(type)$population,
  selection = nsgaControl(type)$selection,
  crossover = nsgaControl(type)$crossover,
  mutation = nsgaControl(type)$mutation,
  popSize = 50,
  nObj = ncol(fitness(matrix(10000, ncol = 100, nrow = 100))),
  pcrossover = 0.8,
  pmutation = 0.1,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  monitor = if (interactive()) nsgaMonitor 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 `nsga_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. See `nsga_Selection()` for available functions.

`crossover` an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See `nsga_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 `nsga_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.

`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 nsgaMonitor 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,
```

```
                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,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

`nsga2-class` *Virtual Class 'nsga2 - Simple Class for subassigment Values'*

Description

The class 'nsga2' is a simple class union (`setClassUnion()`) of 'numeric', 'logical' and 'matrix'.

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).

`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.

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

`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.

Objects from the Class

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

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 = nsgaControl(type)$population,
  selection = nsgaControl(type)$selection,
  crossover = nsgaControl(type)$crossover,
  mutation = nsgaControl(type)$mutation,
  popSize = 50,
  nObj = ncol(fitness(matrix(10000, ncol = 100, nrow = 100))),
  n_partitions,
  pcrossover = 0.8,
  pmutation = 0.1,
  reference_dirs = generate_reference_points,
  maxiter = 100,
  run = maxiter,
  maxFitness = Inf,
  names = NULL,
  suggestions = NULL,
  monitor = if (interactive()) nsgaMonitor 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 nsga_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. See nsga_Selection() for available functions.
crossover	an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See nsga_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 nsga_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.
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 nsgaMonitor 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,
                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,
                n_partitions = 12,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
```

`nsga3-class` *Virtual Class 'nsga3 - Simple Class for subassigment Values'*

Description

The class 'nsga3' is a simple class union (`setClassUnion()`) of 'numeric', 'logical' and 'matrix'.

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).

`popSize` the population size.

`front` Range in which the individual is in the front generated by the function (`non_dominated_fronts()`)

`f` Fronts generated by the function (`non_dominated_fronts()`)

`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.

`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.

`pcrossover` the crossover probability.

`pmutation` the mutation probability.

`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.

`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.

Objects from the Class

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

Examples

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

nsgaControl *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

```
nsgaControl(...)
```

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 = "nsgabin_Population"
  - selection = "nsgabin_tourSelection"
  - crossover = "nsgabin_spCrossover"
  - mutation = "nsgabin_raMutation"
- "real-valued"
  - population = "nsgareal_Population"
  - selection = "nsgareal_tourSelection"
  - crossover = "nsgareal_sbxCrossover"
  - mutation = "nsgareal_polMutation"
- "permutation"
  - population = "nsgaperm_Population"
  - selection = "nsgaperm_tourSelection"
  - crossover = "nsgaperm_oxCrossover"
  - mutation = "nsgaperm_simMutation"
- "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

nsga(), nsga2() and nsga3()

Examples

```
# get and save defaults
defaultControl <- nsgaControl()
print(defaultControl)
# get current defaults only for real-valued search
nsgaControl("real-valued")
# set defaults for selection operator of real-valued search
nsgaControl("real-valued" = list(selection = "nsgareal_lrSelection"))
nsgaControl("real-valued")
# set defaults for selection and crossover operators of real-valued search
nsgaControl("real-valued" = list(selection = "nsgareal_lrSelection",
                                 crossover = "nsgareal_spCrossover"))
nsgaControl("real-valued")
# restore defaults
nsgaControl(defaultControl)
nsgaControl()
```

---

nsgaMonitor *Monitor non-dominated genetic algorithm evolution*

Description

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

Usage

```
nsgaMonitor(object, number_objectives, ...)
```

Arguments

object	an object of class nsga, nsga2 or nsga3, usually resulting from a call to function nsga, nsga2 or nsga3, respectively.
number_objectives	numbers of objective values of the function to evaluate.
...	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, nsgaMonitor 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()

---

nsga_Crossover *Crossover operators in non-dominated genetic algorithms*

Description

Functions implementing crossover non-dominated genetic operator.

Usage

```
nsga_spCrossover(object, parents)

nsgabin_spCrossover(object, parents)

nsgareal_spCrossover(object, parents)
nsgareal_sbxCrossover(object, parents, nc = 20)

nsgaperm_oxCrossover(object, parents)
```

Arguments

object	An object of class "nsga", "nsga2" and "nsga3", usually resulting from a call to function nsga, nsga2 and nsga3.
parents	A two-rows matrix of values indexing the parents from the current population.
nc	Parameters of non-dominated genetic operators.

Value

Return a list with two elements:

children	a matrix of dimension 2 times the number of decision variables containing the generated offsprings;
fitness	a vector of length 2 containing the fitness values for the offsprings. A value NA is returned if an offspring is different (which is usually the case) from the two parents.

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()

---

nsga_Mutation *Mutation operators in non-dominated genetic algorithms*

Description

Functions implementing mutation non-dominated genetic operator.

Usage

```
nsgabin_raMutation(object, parent)

nsgareal_raMutation(object, parent)
nsgareal_polMutation(object, parent, nm = 0.20)

nsgaperm_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.
nm	Parameters of genetic operators.

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.

---

nsga_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

```
nsgabin_Population(object)

nsgareal_Population(object)

nsgaperm_Population(object)
```

Arguments

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

Details

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

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

nsgaperm_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`

---

nsga_Selection *Selection operators in non-dominated genetic algorithms*

Description

Functions implementing selection non-dominated genetic operator.

Usage

```
nsga_lrSelection(object, r, q)
nsga_tourSelection(object, k = 3, ...)

nsgabin_lrSelection(object, r, q)
nsgabin_tourSelection(object, k = 3, ...)

nsgareal_lrSelection(object, r, q)
nsgareal_tourSelection(object, k = 3, ...)

nsgaperm_lrSelection(object, r, q)
nsgaperm_tourSelection(object, k = 3, ...)
```

Arguments

`object`	An object of class `"nsga"`, `"nsga2"` or `"nsga3"`, usually resulting from a call to function `nsga`, `nsga2` or `nsga3`.
`r`	A tuning parameter for the specific selection operator.
`q`	A tuning parameter for the specific selection operator.
`k`	A tuning parameter for the specific selection operator.
`...`	Further arguments passed to or from other methods.

Value

Return a list with two elements:

`population`	a matrix of dimension `object@popSize` times the number of decision variables containing the selected individuals or strings;
`fitness`	a vector of length `object@popSize` containing the fitness values for the selected individuals.

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()`

---

numberOrNAOrMatrix-class

Virtual Class 'numberOrNAOrMatrix - Simple Class for subassigment Values'

Description

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

Objects from the Class

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

Examples

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

pcp *Parallel Coordinate Plots*

Description

The pcp() function for hyperdimensional data visualization, which represents a p-dimensional data point in Cartesian coordinates by a polyline (or curve) intercepting n-parallel axes, where p or the x-axis represents the fitness values and n or the y-axis represents the objectives.

Usage

```
pcp(object)
```

Arguments

object	An object of nsga-class, nsga2-class or nsga3-class. See nsga, nsga2 or nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
#Four Objectives Plotting
dtlz1 <- function (x, nobj = 4){
    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(0,0,0,0),
                upper = c(1,1,1,1),
                popSize = 92,
                n_partitions = 12,
                monitor = FALSE,
                maxiter = 500)
```

```
## End(Not run)
#Not Run
## Not run:
pcp(object = result)

## End(Not run)
```

`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(fitness, reference_points)
```

Arguments

`fitness`	Objective values generated by the algorithm.
`reference_points`	Optimal points to achieve.

Value

A vector with the measurement parameter.

Author(s)

Francisco Benitez

References

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

polar *Polar Area Aiagram*

Description

The `polar()` function is a viable tool for one dimesiona data visualization, which that shows magnitude of a phenomenon as color in two dimension.

Usage

```
polar(fitness)
```

Arguments

`fitness`	An matrix of values representing the fitness of the objective values of nsga-class, nsga2-class or nsga3-class. See nsga, nsga2 or nsga3 for a description of available slots information.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
#Four Objectives Plotting
dtlz1 <- function (x, nobj = 4){
    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 = rep(0,4),
                upper = rep(1,4),
                popSize = 92,
                n_partitions = 12,
                monitor = FALSE,
```

```
                maxiter = 500)

## End(Not run)
#Not Run
## Not run:
polar(fitness = result@fitness)

## End(Not run)
```

`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()`

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()`

scatter *Scatter Plot Functions*

Description

Allows to make scatter plots in publication quality allowing to represent 2-D, 3-D and M-D

Usage

```
scatter(object, ...)
```

Arguments

object	An object of nsga-class, nsga2-class or nsga3-class. See nsga, nsga2 or nsga3 for a description of available slots information.
...	Other arguments passed on to methods. Used to pass the optimal value of the objective function, in case of having it.

Author(s)

Francisco Benitez <benitezfj94@gmail.com>

Examples

```
#Two Objectives Plotting
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,
                n_partitions = 100,
                monitor = FALSE,
                maxiter = 500)

## End(Not run)
#Not run
## Not run:
scatter(object = result)
```

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

`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()`

`Summary` *Summarize non-dominated genetic algorithm evolution*

Description

A function which returns fitness summary and metric measure at each iteration of algorithms search..

Usage

```
Summary(object, ...)
```

Arguments

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

Details

This function records the individuals, objective values and performance metrics generated by iteration for later analysis..

Value

Returns a list with the individuals, objective values and performance metrics by generation.

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.

Bossek, J. (2017). ecr 2.0: A Modular Framework for Evolutionary Computation in R. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17) Companion, Berlin, Germany, 1187–1193.

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