Package 'gatbxr'

November 13, 2026

Title Port of the Sheffield Genetic Algorithm Toolbox for 'MATLAB'

Version 0.2.0

Description Provides the functions of the Genetic Algorithm Toolbox for 'MATLAB' developed at the University of Sheffield (Chipperfield et al. 1994, <https://github.com/UoS-CODeM/GA-Toolbox>), rewritten in R. It includes operators for creating binary, integer and real-valued populations, rank-based and linear fitness assignment, roulette wheel and stochastic universal sampling selection, single-point, multi-point, shuffle and reduced surrogate crossover, discrete, intermediate and line recombination, discrete and breeder genetic algorithm mutation, reinsertion of offspring, and migration between subpopulations.

License GPL (>= 2)

URL https://github.com/drizztxx/gatbxr

BugReports https://github.com/drizztxx/gatbxr/issues

Depends R (>= 3.5.0)

Imports stats

Suggests spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Liang Zhao [aut, cre, trl] (R implementation),
Andrew Chipperfield [aut, cph] (Author of the original 'MATLAB' toolbox),
Peter Fleming [aut, cph] (Author of the original 'MATLAB' toolbox),
Hartmut Pohlheim [aut, cph] (Author of the original 'MATLAB' toolbox),
Carlos Fonseca [aut, cph] (Author of the original 'MATLAB' toolbox),
Alex Shenfield [ctb] (Tested the original 'MATLAB' toolbox),
The University of Sheffield [cph] (Copyright holder of the original 'MATLAB' toolbox)

Maintainer Liang Zhao <wethenwethen@gmail.com>

Repository CRAN

Date/Publication 2026-11-13 22:50:02 UTC

Contents

bs2rv . . . 2
crtbase . . . 4
crtbp . . . 4
crtrp . . . 5
migrate . . . 7
mut . . . 9
mutate . . . 10
mutbga . . . 11
ranking . . . 12
recdis . . . 14
recint . . . 15
reclin . . . 16
recmut . . . 18
recombin . . . 19
reins . . . 20
rws . . . 22
scaling . . . 23
select . . . 24
sus . . . 26
xovmp . . . 26

Index **29**

`bs2rv` *Binary string to real vector*

Description

This function decodes binary chromosomes into vectors of reals. The chromosomes are seen as the concatenation of binary strings of given length, and decoded into real numbers in a specified interval using either standard binary or Gray decoding.

Usage

```
bs2rv(Chrom, FieldD)
```

Arguments

Chrom	matrix containing the chromosomes of the current population. Each row corresponds to one individual's concatenated binary string representation. Leftmost bits are the most significant bits and rightmost are the least significant bits.
FieldD	a list describing how to decode each substring in the chromosome. It must contain the following 7 elements: prec a number giving the length (precision) of each substring; lb, ub lower and upper bounds of each variable; code "binary" or "gray", indicating how each substring is to be decoded; scale "arith" or "log", indicating whether to use arithmetic or logarithmic scaling; lbin, ubin logical values indicating whether or not to include each bound in the representation range. Except for prec, each element may be a single value applied to all variables or a vector with one value per variable.

Value

a real matrix containing the population phenotypes. Each row corresponds to one individual and each column to one variable.

Author(s)

The original 'MATLAB' implementation of bs2rv was written by Carlos Fonseca and updated by Andrew Chipperfield. The R implementation was written by Liang Zhao.

See Also

crtbp

Examples

```
## Create a population of 40 chromosomes of length 400
Chrom = crtbp(Nind = 40, Lind = 400)$Chrom

## Convert to reals in the range [-512, 512] with a precision of 20
## (20 bits represent one real number) on an arithmetic scale by binary decoding.
Phen = bs2rv(Chrom = Chrom,
        FieldD = list(prec=20,lb=-512,ub=512,code="binary",scale="arith",lbin=TRUE,ubin=TRUE))

## Two variables with different bounds, decoded with Gray code
Chrom = crtbp(5, 16)$Chrom
Phen = bs2rv(Chrom,
            FieldD = list(prec=8,lb=c(-1,10),ub=c(1,100),code="gray",
                          scale=c("arith","log"),lbin=TRUE,ubin=TRUE))
```

crtbase *Create base vector*

Description

This function creates a vector containing the base of the loci in a chromosome.

Usage

```
crtbase(Lind, Base = NULL)
```

Arguments

Lind	a number or vector containing the lengths of the alleles. sum(Lind) is the length of the corresponding chromosome.
Base	a number or vector containing the base of the loci contained in the Alleles. Default is set to binary representation.

Value

a vector whose elements correspond to the base of the loci of the associated chromosome structure.

Author(s)

The original 'MATLAB' implementation of crtbase was written by Andrew Chipperfield. The R implementation was written by Liang Zhao.

Examples

```
## 10 binary loci
Base = crtbase(10)

## 3 loci of base 8 followed by 2 loci of base 4
Base = crtbase(c(3,2), c(8,4))
```

crtbp *Create an initial population*

Description

This function creates a binary population of given size and structure.

Usage

```
crtbp(Nind, Lind = NULL, Base = NULL)
```

Arguments

Nind	either a number containing the number of individuals in the new population or a vector of length two containing the number of individuals and their length.
Lind	an optional number containing the length of the individual chromosomes, or a vector containing the base of each locus of the chromosomes.
Base	an optional number containing the base of the chromosome elements or a vector containing the base(s) of the loci of the chromosomes.

Value

a list containing following components:

Chrom	a matrix containing the random valued chromosomes row wise.
Lind	a scalar containing the length of the chromosome.
BaseV	a vector containing the base of the chromosome loci.

Author(s)

The original 'MATLAB' implementation of crtbp was written by Andrew Chipperfield. The R implementation was written by Liang Zhao.

See Also

crtbase, crtrp

Examples

```
## 40 individuals with 10 binary loci
Chrom = crtbp(40,10)$Chrom

## 6 individuals with 5 loci of base 8
Chrom = crtbp(6,5,8)$Chrom

## 6 individuals whose loci have bases 2, 4 and 8
Chrom = crtbp(6,c(2,4,8))$Chrom
```

---

crtrp *Create a real-valued initial population*

Description

This function creates a population of given size of random real-values.

Usage

```
crtrp(Nind, FieldDR)
```

Arguments

Nind	a value containing the number of individuals in the new population.
FieldDR	a matrix of 2 rows by number of variables describing the boundaries of each variable.

Details

The first step in a genetic algorithm is to create an initial population consisting of random individuals. crtrp produces a matrix, Chrom, containing uniformly distributed random values in its elements.

Chrom = crtrp(Nind, FieldDR) creates a random real-valued matrix of size Nind * Nvar, where Nind specifies the number of individuals in the population and Nvar the number of variables of each individual. Nvar is derived from FieldDR with Nvar = NCOL(FieldDR).

FieldDR is a matrix of size 2 * Nvar and contains the boundaries of each variable of an individual. The first row contains the lower bounds, the second row the upper bounds.

FieldDR is also used in other functions such as mutate.

Value

a matrix containing the random valued individuals of the new population of size Nind by number of variables.

Author(s)

The original 'MATLAB' implementation of crtrp was written by Hartmut Pohlheim and tested by Alex Shenfield. The R implementation was written by Liang Zhao.

See Also

mutbga, recdis, recint, reclin

Examples

```
## To create a random population of 6 individuals with 4 variables each:
## Define boundaries on the variables,
FieldDR = matrix(c(-100,-50,-30,-20,
                   100,50,30,20),2,4,byrow=TRUE)
## Create initial population
Chrom = crtrp(6,FieldDR)
```

migrate *Migration of individuals between subpopulations*

Description

This function performs migration of individuals.

Usage

```
migrate(
  Chrom,
  SUBPOP = 1,
  ObjV,
  MIGR = 0.2,
  Select = FALSE,
  Structure = c("net", "neighbourhood", "ring")
)
```

Arguments

Chrom	a matrix containing the individuals of the current population. Each row corresponds to one individual.
SUBPOP	an optional number indicating subpopulations. Default is set to 1 subpopulation.
ObjV	an optional vector containing the objective values (cost values, to be minimized) of the individuals in the current population.
MIGR	a number indicating rate of individuals to be migrated per subpopulation. Default is set to 0.2.
Select	a logical value indicating whether fitness-based selection will be used. Default is set to FALSE. If set to TRUE, then ObjV must also be defined.
Structure	a character string indicating the name of structure of the subpopulations for migration: one of "net" (default, unconstrained migration), "neighbourhood" or "ring".

Details

gatbxr provides support for multiple subpopulations through the use of high-level genetic operator functions and a routine for exchanging individuals between subpopulations. In the literature, the use of multiple populations has been shown, in most cases, to improve the quality of the results obtained using GAs compared to the single population GA.

gatbxr supports the use of a single population divided into a number of subpopulations or $demes$ by modifying the use of data structures such that subpopulations are stored in contiguous blocks within a single matrix.

In this model each subpopulation is evolved over generations by a traditional GA and from time

to time individuals migrate from one subpopulation to another. The amount of migration of individuals and the pattern of that migration determines how much genetic diversity can occur.

`migrate` implements the transfer of individuals between subpopulations, in which `Structure` specifies the population topology over which migration will take place. The default unrestricted migration topology (`net`) is the most general migration strategy. Here, individuals may migrate from any subpopulation to another. For each subpopulation, a pool of potential immigrants is constructed from the other subpopulations. The individual migrants are then determined according to the appropriate selection strategy.

Structure `ring` transfers individuals between directionally adjacent subpopulations. And `neighbourhood` is a similar strategy. Like the `ring` topology, migration is made only between nearest neighbours, however, migration may occur in either direction between subpopulations. For each subpopulation, the possible immigrants are determined, according to the desired selection method, from the adjacent subpopulations and a final selection made from this pool of individuals. This ensures that individuals will not migrate from a subpopulation to the same population.

Value

a list containing following components:

`Chrom`	a matrix containing the individuals of the current population after reinsertion.
`ObjV`	if `ObjV` is given, the vector containing the objective values of the individuals of the current generation after migration; otherwise `NULL`.

Author(s)

The original 'MATLAB' implementation of migrate was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

`select`, `recombin`, `mutate`, `reins`

Examples

```
## create initial population of 5 subpopulations with 10 individuals each
Chrom = crtbp(50,10)$Chrom

## calculate objective values (to be minimized) with sum function
ObjV = apply(Chrom,1,sum)

## choose 20% of the individuals of one subpopulation and replace
## these individuals with the fittest individuals from an adjacent
## subpopulation in a unidirectional ring structure
res = migrate(Chrom, SUBPOP = 5, ObjV, Select = TRUE, Structure = "ring")
```

mut *Discrete mutation (low-level function)*

Description

This function takes the representation of the current population, mutates each element with given probability and returns the resulting population.

Usage

```
mut(OldChrom, Pm = NULL, BaseV = NULL)
```

Arguments

OldChrom	a matrix containing the chromosomes of the current population. Each row corresponds to an individual's string representation.
Pm	a value indicating mutation probability. Default value is Pm = 0.7/Lind, where Lind is the chromosome length.
BaseV	an optional vector of the same length as the chromosome structure defining the base of the individual elements of the chromosome. Default is set to binary representation.

Value

a matrix containing a mutated version of OldChrom.

Author(s)

The original 'MATLAB' implementation of mut was written by Andrew Chipperfield. The R implementation was written by Liang Zhao.

See Also

mutate, mutbga

Examples

```
Chrom = crtbp(40,10)$Chrom

NewChrom = mut(OldChrom=Chrom)
```

mutate *Mutation (high-level function)*

Description

This function takes a matrix `OldChrom` containing the representation of the individuals in the current population, mutates the individuals and returns the resulting population.

The function handles multiple populations and calls the low-level mutation function for the actual mutation process.

Usage

```
mutate(MUT_F = "mut", OldChrom, FieldDR = NULL, MutOpt = NULL, SUBPOP = 1, ...)
```

Arguments

`MUT_F`	character string containing the name of the low-level mutation function, e.g. `"mut"` (default) or `"mutbga"`.
`OldChrom`	matrix containing the chromosomes of the old population. Each line corresponds to one individual.
`FieldDR`	matrix describing the boundaries of each variable (real-values) or defining the base of the variables of each individual (discrete values). Optional for (binary) discrete values
`MutOpt`	an optional vector containing mutation rate and shrink value: 1. MutOpt[1] a number containing the mutation rate in the range $[0, 1]$. By default this value is assumed to $1/Nvar$, where $Nvar$ is the number of variables per individual defined by `NCOL(FieldDR)`. 2. MutOpt[2] a number for shrinking the mutation range in the range $[0, 1]$. By default this value is assumed to be 1 (no shrinking).
`SUBPOP`	an optional number of subpopulations. Default is set to 1.
`...`	other arguments passed on to the mutation function.

Value

a matrix containing the chromosomes of the population after mutation in the same format as `OldChrom`.

Note

`mutate` has the same name as functions in other packages (for example `dplyr::mutate`); use `gatbxr::mutate` if another package masks it.

Author(s)

The original 'MATLAB' implementation of mutate was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

mut, mutbga

Examples

```
## Binary population
Chrom = crtbp(40,10)$Chrom
NewChrom = mutate(MUT_F="mut",OldChrom=Chrom)

## Real-valued population
FieldDR = matrix(c(-100,-50,-30,-20,100,50,30,20),2,4,byrow=TRUE)
Chrom = crtrp(6,FieldDR)
NewChrom = mutate("mutbga",Chrom,FieldDR)
```

---

mutbga *Mutation of real-valued population*

Description

This function implements the mutation operator of the Breeder Genetic Algorithm.

Usage

```
mutbga(OldChrom, FieldDR, MutOpt = NULL)
```

Arguments

OldChrom	a matrix containing the chromosomes of the old population. Each row corresponds to one individual.
FieldDR	a matrix of 2 rows by number of variables describing the boundaries of each variable.
MutOpt	an optional vector containing mutation rate and shrink value: 1. MutOpt[1] a number containing the mutation rate in the range $[0, 1]$. By default this value is assumed to $1/Nvar$, where $Nvar$ is the number of variables per individual defined by NCOL(FieldDR). 2. MutOpt[2] a number for shrinking the mutation range in the range $[0, 1]$. By default this value is assumed to be 1 (no shrinking).

Details

mutbga takes a matrix OldChrom containing the real representation of the individuals in the current population, mutates the individuals with given probability and returns the resulting population.

The mutation of a variable is computed as follows:
mutated variable = variable + $MutMx * range * MutOpt[2] * delta$, where $MutMx$ = 1 or -1 with probability $MutOpt[1]$ (+ or - with equal probability), else 0. $range = 0.5*$ domain of variable (search interval defined by FieldDR). $delta = sum(0tom - 1)pi * 2^{-}i, pi = 1$ with probability

$1/m$, else 0, m = 20.

With $m = 20$, the mutation operator is able to locate the optimum up to a precision of $range * MutOpt[2] * 2^{-}19$.

`mutbga` is able to generate most points in the hypercube defined by the variables of the individual and the range of the mutation. However, it tests more often near the variable, that is, the probability of small step sizes is greater than that of larger step sizes.

Value

a matrix containing the chromosomes of the population after mutation in the same format as `OldChrom`.

Author(s)

The original 'MATLAB' implementation of mutbga was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

References

Muhlenbein, H. and Schlierkamp-Voosen, D. (1993) Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization. *Evolutionary Computation*, **1**(1), 25–49. doi:10.1162/evco.1993.1.1.25

See Also

`mutate`, `recdis`, `recint`, `recmut`, `reclin`

Examples

```
## Initialise a real-valued population
FieldDR = matrix(c(-100,-50,-30,-20,100,50,30,20),2,4,byrow=TRUE)
Chrom = crtrp(6,FieldDR)

## Perform mutation
NewChrom = mutbga(Chrom,FieldDR)
```

---

ranking *Rank-based fitness assignment*

Description

This function ranks individuals represented by their associated cost, to be *minimized*, and returns a vector `FitnV` containing the corresponding individual fitnesses. For multiple subpopulations the ranking is performed separately for each subpopulation.

Usage

```
ranking(ObjV, RFun = c(2, 0), SUBPOP = 1)
```

Arguments

ObjV	a vector containing the objective values of the individuals in the current population (cost values).
RFun	if RFun is a number in $[1, 2]$, linear ranking is assumed and the value indicates the selective pressure. If RFun is a vector of 2 elements, then RFun[1] indicates the selective pressure and RFun[2] indicates the ranking method: when RFun[2] = 0, linear ranking is used; when RFun[2] = 1, non-linear ranking is used. If RFun is a vector with length(RFun) > 2, it contains the fitness to be assigned to each rank. It should have the same length as the number of individuals per subpopulation. Usually RFun is monotonically increasing. If RFun is NULL, linear ranking and a selective pressure of 2 are assumed.
SUBPOP	an optional number indicating subpopulations. Default is set to 1 subpopulation.

Details

Individuals with equal objective values receive the same fitness, namely the average of the fitness values assigned to their ranks.

Value

a vector containing the fitness values of the individuals in the current population.

Author(s)

The original 'MATLAB' implementation of ranking was written by Hartmut Pohlheim and Carlos Fonseca. The R implementation was written by Liang Zhao.

See Also

scaling, select

Examples

```
Chrom = crtbp(40,10)$Chrom

## Objective function is the sum of individuals
ObjV = apply(Chrom,1,sum)

## Linear ranking with selective pressure 2
FitnV = ranking(ObjV)

## Non-linear ranking with selective pressure 3
FitnV = ranking(ObjV, RFun = c(3,1))
```

recdis *Discrete recombination*

Description

This function performs discrete recombination between pairs of individuals and returns the new individuals after mating.

Usage

```
recdis(OldChrom)
```

Arguments

OldChrom	a matrix containing the chromosomes of the old population. Each line corresponds to one individual.

Details

recdis is a function applicable to both populations of real-value variables and binary or integer variables. The pairs are mated in order, odd row with the next even row. If the number of rows in the matrix OldChrom is odd then the last row is not mated and added at the end of NewChrom. The population should therefore be organised into contiguous pairs that require mating. This can be achieved by using the function ranking to assign a fitness level to each individual and a selection function (e.g. select) to select individuals with a probability related to their fitness in the current population.

Value

a matrix containing the chromosomes of the population after mating, ready to be mutated and/or evaluated, in the same format as OldChrom.

Algorithm

Discrete recombination exchanges variable values between the individuals. For each variable the parent who contributes its variable value to the offspring is chosen randomly with equal probability.

Discrete recombination can generate the corners of the hypercube defined by the parents.

Author(s)

The original 'MATLAB' implementation was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

recombin, recint, reclin, ranking, sus, rws

Examples

```
## Initial a real-valued population
FieldDR = matrix(c(-100,-50,-30,-20,100,50,30,20),2,4,byrow=TRUE)
Chrom = crtrp(6,FieldDR)

## Perform discrete recombination
NewChrom = recdis(Chrom)
```

---

recint *Intermediate recombination*

Description

This function performs extended intermediate recombination between pairs of individuals and returns the new individuals after mating.

Usage

```
recint(OldChrom)
```

Arguments

OldChrom	a matrix containing the chromosomes of the old population. Each line corresponds to one individual.

Details

`recint` is a function only applicable to populations of real-value variables (and not binary or integer). The pairs are mated in order, odd row with the next even row. If the number of rows in the matrix `OldChrom` is odd then the last row is not mated and added at the end of `NewChrom`. The population should therefore be organised into contiguous pairs that require mating. This can be achieved by using the function `ranking` to assign a fitness level to each individual and a selection function (e.g. `select`) to select individuals with a probability related to their fitness in the current population.

Value

a matrix containing the chromosomes of the population after mating, ready to be mutated and/or evaluated, in the same format as OldChrom.

Algorithm

Intermediate recombination combines parent values using the following rule:

$$offspring = parent1 + Alpha * (parent2 - parent1)$$

where $Alpha$ is a scaling factor chosen uniformly at random in the interval $[-0.25, 1.25]$. `recint` produces a new Alpha for each pair of values to be combined.

Intermediate recombination can generate any point within a hypercube slightly larger than that defined by the parents.

Intermediate recombination is similar to line recombination `reclin`. Whereas `recint` uses a new Alpha factor for each pair of values combined together, `reclin` uses one Alpha factor for each pair of parents.

Author(s)

The original 'MATLAB' implementation was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

`recombin`, `recdis`, `reclin`, `ranking`, `sus`, `rws`

Examples

```
## Initial a real-valued population
FieldDR = matrix(c(-100,-50,-30,-20,100,50,30,20),2,4,byrow=TRUE)
Chrom = crtrp(6,FieldDR)

## Perform intermediate recombination
NewChrom = recint(Chrom)
```

---

reclin *Line recombination*

Description

This function performs line recombination between pairs of individuals and returns the new individuals after mating.

Usage

```
reclin(OldChrom)
```

Arguments

`OldChrom`	a matrix containing the chromosomes of the old population. Each line corresponds to one individual.

Details

reclin is a function only applicable to populations of real-value variables (and not binary or integer). The pairs are mated in order, odd row with the next even row. If the number of rows in the matrix OldChrom is odd then the last row is not mated and added at the end of NewChrom. The population should therefore be organised into contiguous pairs that require mating. This can be achieved by using the function ranking to assign a fitness level to each individual and a selection function (e.g. select) to select individuals with a probability related to their fitness in the current population.

Value

a matrix containing the chromosomes of the population after mating, ready to be mutated and/or evaluated, in the same format as OldChrom.

Algorithm

Line recombination combines parent values using the following rule:

$$offspring = parent1 + Alpha * (parent2 - parent1)$$

where $Alpha$ is a scaling factor chosen uniformly at random in the interval $[-0.25, 1.25]$. reclin produces one Alpha factor for each pair of parents combined.

Line recombination can generate any point on a slightly longer line than that defined by the parents.

Line recombination is similar to intermediate recombination recint. Whereas reclin uses one Alpha factor for each pair of parents combined together, recint uses a new Alpha factor for each pair of values.

Author(s)

The original 'MATLAB' implementation was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

recombin, recdis, recint, ranking, sus, rws

Examples

```
## Initial a real-valued population
FieldDR = matrix(c(-100,-50,-30,-20,100,50,30,20),2,4,byrow=TRUE)
Chrom = crtrp(6,FieldDR)

## Perform line recombination
NewChrom = reclin(Chrom)
```

recmut *Line recombination with mutation features*

Description

This function performs line recombination with mutation features between pairs of individuals.

Usage

```
recmut(OldChrom, FieldDR, MutOpt = NULL)
```

Arguments

OldChrom	matrix containing the chromosomes of the old population. Each line corresponds to one individual.
FieldDR	matrix describing the boundaries of each variable (real-values) or defining the base of the variables of each individual (discrete values). Optional for (binary) discrete values
MutOpt	an optional vector containing recombination rate and shrink value: 1. MutOpt[1] a number containing the recombination rate in the range $[0, 1]$. By default this value is set to 1. 2. MutOpt[2] a number for shrinking the mutation range in the range $[0, 1]$. By default this value is assumed to be 1 (no shrinking).

Details

recmut is a function only applicable to populations of real-value variables (and not binary or integer). The pairs are mated in order, odd row with the next even row. If the number of rows in the matrix OldChrom is odd then the last row is not mated and added at the end of NewChrom. The population should therefore be organised into contiguous pairs that require mating. This can be achieved by using the function ranking to assign a fitness level to each individual and a selection function (e.g. select) to select individuals with a probability related to their fitness in the current population.

The offspring of a pair of two parents are computed as follows:
$offspring1 = parent1 + RecMx * range * MutOpt[2] * delta * Diff$
$offspring2 = parent2 + RecMx * range * MutOpt[2] * delta * -Diff$
where,
$RecMx$ = 1 or -1 with probability $MutOpt[1]$ (- with probability 0.9), else 0.
$range = 0.5*$ domain of variable (search interval defined by FieldDR).
$delta = sum(0tom - 1)pi * 2^{-}i, pi = 1$ with probability $1/m$, else 0, m = 20.
$Diff = (parent2 - parent1)/|norm(parent2) - norm(parent1)|$. Pairs of parents with equal norms are not moved.

The recombination operator recmut generates offspring in a direction defined by the parents (line recombination). It tests more often outside the area defined by the parents and in the direction of parent1. The point for the offspring is defined by features of the mutation operator. The probability of small step sizes is greater than that of bigger steps (see mutbga).

Value

a matrix containing the chromosomes of the population after mating, ready to be mutated and/or evaluated, in the same format as OldChrom.

Note

This function doesn't work with high level recombination function `recombin`

Author(s)

The original 'MATLAB' implementation of recmut was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

`mutate`, `mutbga`, `reclin`

Examples

```
## Initial a real-valued population
FieldDR = matrix(c(-100,-50,-30,-20,100,50,30,20),2,4,byrow=TRUE)
Chrom = crtrp(6,FieldDR)

## Perform line recombination with mutation
NewChrom = recmut(Chrom,FieldDR)
```

---

recombin *Recombination (high-level function)*

Description

This function performs recombination between pairs of individuals and returns the new individuals after mating. The function handles multiple populations and calls the low-level recombination function for the actual recombination process.

Usage

```
recombin(REC_F, Chrom, RecOpt = 0.7, SUBPOP = 1, ...)
```

Arguments

`REC_F`	character string containing the name of the recombination or crossover function, e.g. `"xovsp"` or `"recdis"`.
`Chrom`	a matrix containing the chromosomes of the old population. Each row corresponds to one individual.
`RecOpt`	an optional value containing the probability of recombination/crossover occurring between pairs of individuals. It is only used by the crossover functions (`xov*`). Default is set to 0.7.

SUBPOP	an optional number indicating subpopulations. Default is set to 1.
...	other arguments passed on to the crossover function.

Value

a matrix containing the chromosomes of the population after recombination in the same format as Chrom.

Note

This function doesn't work with low level recombination function recmut.

Author(s)

The original 'MATLAB' implementation of recombin was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

xovmp, recdis, recint, reclin

Examples

```
Selch = crtbp(40,10)$Chrom

NewChrom = recombin(REC_F="xovsp",Chrom=Selch)
```

---

reins *Reinsertion of offspring in population replacing parents*

Description

This function reinserts offspring in the population.

Usage

```
reins(
  Chrom,
  SelCh,
  SUBPOP = 1,
  InsOpt = c(0, 1),
  ObjVCh = NULL,
  ObjVSel = NULL
)
```

Arguments

Chrom	matrix containing the individuals (parents) of the current population. Each row corresponds to one individual.
SelCh	matrix containing the offspring of the current population. Each row corresponds to one individual.
SUBPOP	an optional number indicating subpopulations. Default is set to 1 subpopulation.
InsOpt	an optional vector containing the insertion method parameters. InsOpt[1] is a number indicating the kind of insertion: 0 stands for uniform insertion and 1 for fitness-based insertion. If omitted, 0 is assumed. InsOpt[2] is the rate of offspring to be inserted per subpopulation. If omitted, 1.0 is assumed.
ObjVCh	an optional vector containing the objective values of the individuals (parents, Chrom) in the current population, needed for fitness-based insertion. It saves recalculation of objective values for the population.
ObjVSel	an optional vector containing the objective values of the offspring (SelCh) in the current population, needed for partial insertion of offspring. It saves recalculation of objective values for the population.

Value

a list containing following components:

ObjVCh	if ObjVCh is given, the vector containing the objective values of the individuals of the current generation after reinsertion (updated only if ObjVSel is also given); otherwise NULL.
Chrom	matrix containing the individuals of the current population after reinsertion.

Author(s)

The original 'MATLAB' implementation of reins was written by Hartmut Pohlheim and tested by Alex Shenfield. The R implementation was written by Liang Zhao.

See Also

select, recombin, mutate

Examples

```
## Parents and their objective values (to be minimized)
Chrom = crtbp(10,8)$Chrom
ObjVCh = apply(Chrom,1,sum)

## Offspring and their objective values
SelCh = crtbp(6,8)$Chrom
ObjVSel = apply(SelCh,1,sum)

## Replace the worst parents with all offspring (fitness-based insertion)
res = reins(Chrom, SelCh, InsOpt = c(1,1), ObjVCh = ObjVCh, ObjVSel = ObjVSel)
```

```
## Insert the best 50% of the offspring at uniformly random positions
res = reins(Chrom, SelCh, InsOpt = c(0,0.5), ObjVSel = ObjVSel)
```

---

rws *Roulette Wheel Selection*

Description

This function selects a given number of individuals from a population.

Usage

```
rws(FitnV, Nsel)
```

Arguments

FitnV	a vector containing the fitness values of the individuals in the population.
Nsel	a number indicating how many individuals are to be selected.

Details

rws probabilistically selects Nsel individuals for reproduction according to their fitness, FitnV, in the current population.

NewChrIx = rws(FitnV,Nsel) selects Nsel individuals from a population using roulette wheel selection. FitnV is a vector containing a performance measure for each individual in the population. This can be achieved by using the function ranking or scaling to assign a fitness level to each individual.

rws is a low-level selection function normally called by select.

Value

a vector containing the indexes of the selected individuals relative to the original population, shuffled. The new population, ready for mating, can be obtained by calculating OldChrom[NewChrIx,].

Algorithm

A form of roulette wheel selection is implemented by obtaining a cumulative sum of the fitness vector, FitnV, and generating Nsel uniformly at random distributed numbers between 0 and sum(FitnV). The index of the individuals selected is determined by comparing the generated numbers with the cumulative sum vector. The probability of an individual being selected is then given by:

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{Nind} f(x_i)}$$

where $f(x_i)$ is the fitness of individual x_i and $F(x_i)$ is the probability of that individual being selected.

Author(s)

The original 'MATLAB' implementation of rws was written by Carlos Fonseca and Andrew Chipperfield. The R implementation was written by Liang Zhao.

See Also

select, sus, reins, ranking, scaling

Examples

```
## Consider a population of 8 individuals with the assigned fitness values, FitnV:
FitnV = c(1.5,1.35,1.21,1.07,0.92,0.78,0.64,0.5)
## Select the indices of 6 individuals:
NewChrIx = rws(FitnV,6)
```

---

scaling *Linear fitness scaling*

Description

This function implements a linear fitness scaling algorithm. For multiple subpopulations scaling is performed separately for each subpopulation.

Usage

```
scaling(ObjV, Smul = 2, SUBPOP = 1)
```

Arguments

ObjV	a vector containing the non-negative objective values of the individuals.
Smul	an optional value used to determine the upper bound, default is set to 2.
SUBPOP	an optional value indicating subpopulations. Default is set to 1 subpopulation.

Details

scaling converts the objective values, ObjV, into a fitness measure with a known upper bound, determined by the value of Smul, such that,

$$F(x_i) = \alpha f(x_i) + b$$

where $f(x_i)$ is the objective value of individual x_i, α is a scaling coefficient, b is an offset and $F(x_i)$ is the resulting fitness value of individual x_i. If f_{ave} is the average objective value in the current generation, then the maximum fitness of the scaled population is upper bounded at $f_{ave} * Smul$. If Smul is omitted then the default value is set to 2. The average fitness of the scaled population is also set to f_{ave}.

If the upper bound cannot be reached without producing negative fitness values, scaling instead chooses the offset b such that the smallest scaled fitness value is zero. If all objective values of a subpopulation are equal, every individual is assigned the average value.

Value

a vector containing the individual fitnesses for the current population.

Note

Negative objective values are not supported, as linear scaling becomes unreliable in that case. ranking is generally recommended instead; scaling is included only for the sake of completeness.

Author(s)

The original 'MATLAB' implementation of scaling was written by Andrew Chipperfield. The R implementation was written by Liang Zhao.

References

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

See Also

ranking, reins, rws, select, sus

Examples

```
Chrom = crtbp(40,10)$Chrom

## Objective function is the sum of individuals
ObjV = apply(Chrom,1,sum)
FitnV = scaling(ObjV)
```

---

select *Selection (high-level function)*

Description

This function performs universal selection. The function handles multiple populations and calls the low level selection function for the actual selection process.

Usage

```
select(SEL_F = "sus", Chrom, FitnV, GGAP = 1, SUBPOP = 1, ...)
```

Arguments

SEL_F	character string containing the name of the low-level selection function, e.g. "sus" (default) or "rws".
Chrom	a matrix containing the individuals (parents) of the current population. Each row corresponds to one individual.
FitnV	a vector containing the fitness values of the individuals in the population.
GGAP	an optional number indicating rate of individuals to be selected. Default is set to 1.
SUBPOP	an optional number indicating subpopulations. Default is set to 1 subpopulation.
...	other arguments passed on to the selection function.

Value

a matrix containing the selected individuals.

Note

`select` has the same name as functions in other packages (for example `dplyr::select`); use `gatbxr::select` if another package masks it.

Author(s)

The original 'MATLAB' implementation of select was written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

sus, rws, ranking

Examples

```
Chrom = crtbp(40,10)$Chrom

## Objective function is the sum of individuals
FitnV = ranking(apply(Chrom,1,sum))

Selch = select(SEL_F="sus",Chrom=Chrom,FitnV=FitnV,GGAP=0.9)
```

sus *Stochastic Universal Sampling*

Description

This function performs selection with stochastic universal sampling.

Usage

```
sus(FitnV, Nsel)
```

Arguments

FitnV	a vector containing the fitness values of the individuals in the population.
Nsel	a number indicating how many individuals are to be selected.

Value

a vector containing the indexes of the selected individuals relative to the original population, shuffled. The new population, ready for mating, can be obtained by calculating OldChrom[NewChrIx,].

Author(s)

The original 'MATLAB' implementation of sus was written by Hartmut Pohlheim and Carlos Fonseca. The R implementation was written by Liang Zhao.

Examples

```
Chrom = crtbp(40,10)$Chrom

## Objective function is the sum of individuals
FitnV = ranking(apply(Chrom,1,sum))

Selch = sus(FitnV=FitnV,Nsel=2)
```

xovmp *Crossover operators for binary representation*

Description

These functions take a matrix OldChrom containing the binary representation of the individuals in the current population, apply crossover to consecutive pairs of individuals with defined probability and return the resulting population.

Usage

```
xovmp(OldChrom, Px = 0.7, Npt = 0, Rs = FALSE)

xovsp(OldChrom, Px = 0.7)

xovdp(OldChrom, Px = 0.7)

xovsh(OldChrom, Px = 0.7)

xovsprs(OldChrom, Px = 0.7)

xovdprs(OldChrom, Px = 0.7)

xovshrs(OldChrom, Px = 0.7)
```

Arguments

OldChrom	a matrix containing the chromosomes of the old population. Each row corresponds to one individual.
Px	a number indicating the probability of recombination occurring between pairs of individuals. If omitted, 0.7 is used.
Npt	an optional number indicating the number of crossover points to use. Specifically, 1 indicates single point recombination; 2 indicates double point recombination; and 0 indicates shuffle point recombination. Default is set to 0.
Rs	an optional logical value indicating whether or not to force the production of offspring different from their parents. Default is set to FALSE.

Details

xovmp is the general crossover function for binary representation and is called by all other crossover functions. When xovmp is called by the high-level function recombin, its additional arguments can be passed through .... With different combinations of arguments, it can be used identically to xovsp, xovdp, xovsh, xovsprs, xovdprs and xovshrs. Although these functions are redundant, they are kept for completeness.

xovsp performs single-point crossover between pairs of individuals. When called by recombin, recombin("xovsp",Chrom) is identical to recombin("xovmp",Chrom,Npt=1).

xovdp performs double-point crossover between pairs of individuals. When called by recombin, recombin("xovdp",Chrom) is identical to recombin("xovmp",Chrom,Npt=2).

xovsh performs shuffle-point crossover between pairs of individuals. When called by recombin, recombin("xovsh",Chrom) is identical to recombin("xovmp",Chrom).

xovsprs performs single-point reduced surrogate crossover between pairs of individuals. When called by recombin, recombin("xovsprs",Chrom) is identical to recombin("xovmp",Chrom,Npt=1,Rs=TRUE).

`xovdprs` performs double-point reduced surrogate crossover between pairs of individuals. When called by `recombin`, `recombin("xovdprs",Chrom)` is identical to `recombin("xovmp",Chrom,Npt=2,Rs=TRUE)`.

`xovshrs` performs shuffle-point reduced surrogate crossover between pairs of individuals. When called by `recombin`, `recombin("xovshrs",Chrom)` is identical to `recombin("xovmp",Chrom,Rs=TRUE)`.

Value

a matrix containing the offspring after mating, ready to be mutated and/or evaluated, in the same format as OldChrom.

Author(s)

The original 'MATLAB' implementation of xovmp was written by Carlos Fonseca and updated by Andrew Chipperfield; the other crossover functions were written by Hartmut Pohlheim. The R implementation was written by Liang Zhao.

See Also

`recombin`

Examples

```
Chrom = crtbp(40,10)$Chrom

## Shuffle crossover
Selch = xovmp(OldChrom=Chrom)

## Single-point crossover
Selch = xovsp(Chrom, Px = 0.8)
```

Index

bs2rv, 2

crtbase, 4, *5*
crtbp, *3*, 4
crtrp, *5*, 5

migrate, 7
mut, 9, *11*
mutate, *6*, *8*, *9*, 10, *12*, *19*, *21*
mutbga, *6*, *9*, *11*, 11, *18*, *19*

ranking, 12, *14–18*, *22–25*
recdis, *6*, *12*, 14, *16*, *17*, *20*
recint, *6*, *12*, *14*, 15, *17*, *20*
reclin, *6*, *12*, *14*, *16*, 16, *19*, *20*
recmut, *12*, 18, *20*
recombin, *8*, *14*, *16*, *17*, *19*, 19, *21*, *27*, *28*
reins, *8*, 20, *23*, *24*
rws, *14*, *16*, *17*, 22, *24*, *25*

scaling, *13*, *22*, *23*, 23
select, *8*, *13–15*, *17*, *18*, *21–24*, 24
sus, *14*, *16*, *17*, *23–25*, 26

xovdp *(*xovmp*)*, 26
xovdprs *(*xovmp*)*, 26
xovmp, *20*, 26
xovsh *(*xovmp*)*, 26
xovshrs *(*xovmp*)*, 26
xovsp *(*xovmp*)*, 26
xovsprs *(*xovmp*)*, 26