Package 'adana'

October 12, 2022

Title Adaptive Nature-Inspired Algorithms for Hybrid Genetic Optimization

Version 1.1.0

Date 2022-02-01

Author Zeynel Cebeci [aut, cre], Erkut Tekeli [aut], Cagatay Cebeci [aut]

Maintainer Erkut Tekeli <etekeli@atu.edu.tr>

Description The Genetic Algorithm (GA) is a type of optimization method of Evolutionary Algorithms. It uses the biologically inspired operators such as mutation, crossover, selection and replacement.Because of their global search and robustness abilities, GAs have been widely utilized in machine learning, expert systems, data science, engineering, life sciences and many other areas of research and business. However, the regular GAs need the techniques to improve their efficiency in computing time and performance in finding global optimum using some adaptation and hybridization strategies. The adaptive GAs (AGA) increase the convergence speed and success of regular GAs by setting the parameters crossover and mutation probabilities dynamically. The hybrid GAs combine the exploration strength of a stochastic GAs with the exact convergence ability of any type of deterministic local search algorithms such as simulated-annealing, in addition to other nature-inspired algorithms such as ant colony optimization, particle swarm optimization etc. The package 'adana' includes a rich working environment with its many functions that make possible to build and work regular GA, adaptive GA, hybrid GA and hybrid adaptive GA for any kind of optimization problems. Cebeci, Z. (2021, ISBN: 9786254397448).

Depends R (>= 4.0.0)

License GPL-3

LazyData false

Imports stats, optimx, ROI, ROI.plugin.optimx

NeedsCompilation no

Repository CRAN

Date/Publication 2022-02-23 19:50:02 UTC

R topics documented:

adana-package *Adaptive Nature-inspired Algorithms for Hybrid Genetic Optimization*

Description

The package **adana** provides the functions related to the hyrid adaptive genetic algorithms for solving optimization problems.

Details

The Genetic Algorithm (GA) is a type of optimization method of Evolutionary Algorithms. It uses the biologically inspired operators such as mutation, crossover, selection and replacement.Because of their global search and robustness abilities, GAs have been widely utilized in machine learning, expert systems, data science, engineering, life sciences and many other areas of research and business. However, the regular GAs need the techniques to improve their efficiences in computing time and performance in finding global optimum using some adaptation and hybridization strategies. The adaptive GAs (AGA) increase the convergence speed and success of regular GAs by setting the parameters crossover and mutation probabilities dynamically. The hybrid GAs combine the exploration strength of a stochastic GAs with the exact convergence ability of any type of deterministic local search algorithms such as simulated-annealing, in addition to other nature-inspired algorithms such as ant colony optimization, particle swarm optimization etc. The package 'adana' includes a rich working environment with its many functions that make possible to build and work regular GA, adaptive GA, hybrid GA and hybrid adaptive GA for any kind of optimization problems.

Author(s)

Zeynel Cebeci, Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari [Genetic Algorithms and Optimization Applications with R], 535 p. Nobel Academic Publishing. Ankara

See Also

adana

adana *Adaptive Nature-inspired Algorithms for Hybrid Genetic Optimization*

Description

The adana function is a GA function that can be used for any single-objective optimization problem.

Usage

```
adana(gatype = "gga", objective = "max", maxiter = 100, initfunc = initbin, fitfunc,
      selfunc = seltour, crossfunc = px1, mutfunc = bitmut, replace = elitism,
      adapfunc = NULL, hgafunc = NULL, monitorfunc = NULL, n = 100, m = 8,
      lb = rep(0, 8), ub = rep(1, 8), nmode = "real", type = 1, permset = 0:9,
     prevpop = NULL, selt = 2, selbc = 0.5, selc = 2, selk = 1.005, sells = 1.5,
     selns = 0.5, selps = 0.5, sels = 1.5, selt0 = 50, selw = 2, reptype = FALSE,
     cxpc = 0.9, cxpc2 = 0.8, cxon = 2, cxk = 2, cxps = 0.5, cxa = 0, cxb = 0.15,
     cxealfa = 1, cxalfa = 0.5, mutpm = 0.05, mutpm2 = 0.2, mutb = 2, mutpow = 2,
    mutq = 0.5, mutmy = c(), mutsdy = c(), adapa = 0.75, adapb = 0.5, adapc = 0.1,
     adapd = 0.1, hgastep = 10, hgans = 1, hgaftype = "w", reps = 1, repk = 10,
     lambda = 1, tercrit = c(1), abdif = 1e-06, bestdif = 1e-06, objval = 0,
    optdif = 1e-06, rmcnt = 10, rmdif = 1e-06, phidif = 1e-06, rangedif = 1e-06,
     meandif = 1e-06, sddif = 1e-06, mincv = 0.001, simlev = 0.95, maxtime = 60,
      keepbest = TRUE, parfile = NULL, verbose = FALSE, ...)
```

Arguments

gatype	Type of GA. • "gga": generational GA • "ssga": steady state GA Default value is "gga"
objective	Type of optimization. • "min": minimization • "max": maximization Default value is "max"
maxiter	Maximum generation number Default value is 100
initfunc	Name of fitness function. • "initbin" : for binary-coded • "initval" : for value-coded • "initperm": for permutation-coded • or user-defined initialize function Default value is "initbin"
fitfunc	Fitness function
selfunc	Name of the selection function Default value is "seltour"
crossfunc	Name of the crossover function Default value is "px1"
mutfunc	Name of the mutation function Default value is "bitmut"
replace	Name of the replacement function Default value is "elitism"

adapfunc	Name of the adaptation function
hgafunc	The name of the function that will do the hybridization.
monitorfunc	Monitoring function
n	Population size
m	Length of chromosome
lb	A vector containing lower bounds for variables in value-coded problems
ub	A vector containing upper bounds for variables in value-coded problems
nmode	Value mode for initiating value-coding problems • "integer" • "real" Default value is "real"
type	Integer indicating the type of initialization matrix. • 1 : contains the chromosome, fitness value and generation number. • 2 : contains only the chromosome.
permset	A vector containing the ordinal values to be used in permutation-coded initialization. Default value is 0:9
prevpop	A matrix containing previously prepared chromosomes during initialization.
selt	The tournament size for the seltour and seltour2 functions. Default value is 2.
selbc	The base parameter for the selers function. Default value is 0.5.
selc	Scale parameter for selsscale and selsscale2 functions Default value is 2.
selk	Power parameter for selpscale function Default value is 1.005.
sells	Scale parameter for sellscale function Default value is 1.5.
selns	Polynomal coefficient for selnlrs function Default value is 0.5.
selps	Cut-point threshold value for seltrunc function. Default value is 0.5.
sels	Selection pressure for sellrs, sellrs3 and selrscale2 functions. Default value is 1.5.
selt0	The starting temperature for selboltour function. Default value is 50.
selw	Window size for selwscale function. Default value is 2.

reptype	TRUE value is entered for the sampling with replacement when the seltour and seltour2 are used. Default value is FALSE.
cxpc	Crossover ratio Default value is 0.9.
cxpc2	Adapted crossover ratio for the leitingzhi function.
cxon	Number of offspring per mating in crossover. Default value is 2.
cxk	Number of cut-points for multi-point crossover. Default value is 2.
cxps	Probability threshold value for hux, ux, ux2, dc crossovers. Default value is 0.5.
cxa	Location Parameter for lapx crossover Default value is 0.
cxb	Scale Parameter for lapx crossover Default value is 0.15
cxealfa	The random alpha value for sax, wax, ebx crosses. It is determined dynamically, but for some controlled studies, a fixed value can be assigned.
cxalfa	The random alpha value for sax, wax, ebx crosses. It is determined dynamically, but for some controlled studies, a fixed value can be assigned.
mutpm	Mutation rate Default value is 0.05
mutpm2	Adaptive mutation rate
mutb	The exponent value used to avoid uniformity in unimut and nunimut2. Default value is 0.5
mutpow	The exponent value for powmut and powmut2 functions. Default value is 2.
mutq	A number. Value of q for bsearchmut1 Default value is 0.5
mutmy	A vector. Vector of means of genes
mutsdy	A vector. Vector of standard deviations of genes
adapa	Adaptation threshold a for leitingzhi function Default value is 0.5
adapb	Adaptation threshold b for leitingzhi function Default value is 0.75
adapc	Crossover adaptation threshold for adana3 function Default value is 0.2
adapd	Mutation adaptation threshold for adana3 function Default value is 0.05

`hgastep` In a hybrid GA implementation, it is an integer indicating how many generations the hybrid optimization algorithm will be called.
Default value is 10

`hgans` Number of individuals to be transferred to the Optim.
Default value is 2

`hgaftype` Types of fitness to transfer.

- w: individuals with the worst fitness value
- b: individuals with the best fitness value
- r: randomly selected individuals

Default value is "w"

`reps` The number of the best individuals to be selected when elitism is applied.
Default value is 1.

`repk` The selection pressure parameter for the Round Robin function.
Default value is 10.

`lambda` Total number of offspring in replacement algorithms in steady-state replacement type GAs
Default value is 1.

`tercrit` A vector for termination criterion
Default value is (1,13).

`abdif` It is an approach difference value used by the termination criterion 6.
Default value is 1e-06.

`bestdif` The approach value to the global optimum value.
Default value is 1e-06.

`objval` Global optimum. Used by some termination criterion. This criterion is used if the global optimum of the problem is known.

`optdif` It is an approach difference value used by the termination criterion 3.
Default value is 1e-06.

`rmcnt` k value used by the termination criterion 5.
Default value is 10.

`rmdif` It is an approach difference value used by the termination criterion 5.
Default value is 1e-06.

`phidif` It is an approach difference value used by the termination criterion 10.
Default value is 1e-06.

`rangedif` It is an approach difference value used by the termination criterion 8.
Default value is 1e-06.

`meandif` It is an approach difference value used by the termination criterion 4.
Default value is 1e-06.

`sddif` It is an approach difference value used by the termination criterion 7.
Default value is 1e-06.

mincv	Minimum coefficient of variance used by the termination criterion 9. Default value is 0.001.
simlev	It is an approach difference value used by the termination criterion 7. Default value is 0.99.
maxtime	It is maximum runtime (minute) value used by the termination criterion 12. Default value is 60.
keepbest	If the keepbest parameter is TRUE, the best solution value, chromosome, and generation number are saved in the list named bestsol. Default value is TRUE.
parfile	The name of the file where the parameter values are defined if any. Default value is NULL.
verbose	TRUE is assigned to display the statistics of fitness values obtained at the end of the loop in the GA study. Default value is FALSE.
...	additional arguments to be passed to the initialize, select, cross, mutate, hgafunc, and show functions.

Details

adana function is a genetic algorithm function that can be used for all kinds of single-objective optimization problems. The adana function, unlike other GA packages, not only has adaptive GA functions, but also offers specially developed deterministic and self-adaptive techniques called adana1, adana2, and adana3, and is easily hybridized with other optimization methods inspired by nature. Besides, the adana function supports the use of monitors to monitor the progress of the GA run. In addition to containing many crossover and mutation operators, it is coded with a plug-and-play approach so that the user can add custom operator functions that he has developed.

The initialization population is created by using the name of the initialization function and other parameters passed to initfunc with the initialize function. The fit values of individuals in the population are calculated using fitfunc, which is passed to the evaluate function before each iteration. Then, the termination conditions are checked according to the criteria specified in the termination criteria (tercrit) argument via the terminate function. When the termination condition is not met, adana GA continues to run and searches for the best solution. If the keepbest argument is TRUE, the best solution value, chromosome, and generation number are saved in the list named bestsol.

Since the adana function allows adaptive GA studies, from which it is named, it runs a function that is passed with the adapfunc argument and contains the code of the adaptation algorithm. This adaptation function returns the crossover and mutation rates by recalculating. Thus, it strengthens the GA with its exploitation and exploration adaptations.

In order to determine the individuals to be selected for the mating pool, the selection process is done with the selfunc selection function transferred to the select function.

The crossover of selected individuals is executed with the crossover operator in the crossfunc argument passed to the cross function.

Mutation operations are performed with the mutation operator function assigned to mutfunc by the mutate function.

Finally, for GA, replacement is performed by passing the parent population and the offspring population to the replace function.

Hybridization with other optimization techniques can also be done before an iteration of the GA is complete. For this, the hybridization function passed to the hgafunc argument is used. Other parameters that need the optimization technique called in the hybridization function call, are also passed.

Progress made in each GA generation can be monitored visually with a monitor function. For this, the monitor function passed to the monitorfunc argument is used.

In GA implementations, if the required parameters for R functions that perform selection, crossover, mutation, renewal, and other operations are not entered in the function call, their default values are used. The user can change these parameters during the function call to suit the problem. However, there are many parameters used by the adana function and the functions it calls. It may be more practical to use the parameters by saving them to a file. The parfile argument can be used for this purpose.

Value

genfits	A matrix containing statistics for generations.
initpop	A matrix containing the initial population
finalpop	A matrix containing the final population
bestsol	Value of the best solution
objective	Objective of the optimization, *min* or *max*
tcode	Termination code

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

GA Operators: initialize, evaluate, terminate, select, cross, mutate

Initialize Functions: initbin, initval, initperm, initnorm

Selection Functions: selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, selnlrs, selers, seltrunc

Crossover Functions: px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Mutation Functions: bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Replacement Functions: grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrmup1, ssrgenitor, ssrfamtour, ssrx

Adaptation Functions: fixpcmut, ilmdhc, adana1, adana2, adana3, leitingzhi

Hybridization Functions: hgaoptim, hgaoptimx, hgaroi

Examples

```
# Preparing data
material = c("knife", "tin", "potato", "coffee", "sleeping bag", "rope", "compass")
weigth = c(1, 5, 10, 1, 7, 5, 1)
point = c(10, 20, 15, 2, 30, 10, 30)
kspdata = data.frame(material, weigth, point)
capacity = 100

# Fitness Function
kspfit2 = function(x, ...) {
  tpoint = x %*% kspdata$point
  tweigth = x %*% kspdata$weigth
  G1 = tweigth-capacity
  fitval = tpoint-max(0,G1)^2
  return(fitval)
}

# Run GA
n = 20
m = nrow(kspdata)
niter = 100
kspGGA = adana(n=n, m=m, maxiter=niter, objective="max", gatype="gga",
  initfunc=initbin, fitfunc=kspfit2, selfunc=seltour,
  crossfunc=kpx, mutfunc=bitmut, replace=elitism,
  selt=2, reps=4, repk=5, cxon=2, cxk=3, cxpc=0.8,
  mutpm=0.05, tercrit=c(1), keepbest=TRUE,
  verbose=TRUE, monitorfunc=monprogress)

head(kspGGA$finalpop)        # Display Final Population
head(kspGGA$genfits)         # Display Fitness Values According to Generations
bestsol(kspGGA)              # Display Best Solution
kspdata[kspGGA$bestsol$chromosome == 1, ]    # Display Best Chromosome
```

---

adana1 *Adaptive Dynamic Algorithm (Adana 1)*

Description

Adana-1 is an adaptation function that calculates the mutation rates to be applied in generations by sine wave modeling (Cebeci, 2021).

Usage

```
adana1(g, gmax, ...)
```

Arguments

g	Current generation
gmax	Maximum generation
...	Further arguments passed to or from other methods.

Value

pc	Crossover rate
pm	Mutation rate

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

fixpcmut, ilmdhc, adana2, leitingzhi, adana3

Examples

```
gmax <- 1000
g <- c(1, 10, 50, 100, 250, 500, 750, gmax)
adana1(g=g, gmax=gmax)
```

---

adana2 *Adaptive Dynamic Algorithm (Adana 2)*

Description

Adana-2 is an adaptation function that calculates the mutation rates to be applied in generations by square root modeling (Cebeci, 2021).

Usage

```
adana2(g, gmax, ...)
```

Arguments

g	Current generation
gmax	Maximum generation
...	Further arguments passed to or from other methods.

Value

pc	Crossover rate
pm	Mutation rate

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

fixpcmut, ilmdhc, adana1, leitingzhi, adana3

Examples

```
gmax <- 1000
g <- c(1, 10, 50, 100, 250, 500, 750, gmax)
adana1(g=g, gmax=gmax)
```

---

adana3 *Dynamic mutation and crossover function (Adana 3)*

Description

This adaptation function proposed by Cebeci (2021) is an adaptation function that takes into account the cooperation of individuals.

Usage

```
adana3(fitvals, g, gmax, cxpc, mutpm,
                  adapc, adapd, ...)
```

Arguments

fitvals	A vector. Fitness values of current generation
g	Current generation
gmax	Maximum generation
cxpc	Crossover rate. 0 <= cxpc <= 1
mutpm	Mutation rate. 0 <= mutpm <= 1
adapc	Adaptation threshold for crossover rate. 0 <= adapa <= 1. default is 0.05
adapd	Adaptation threshold for mutation rate. 0 <= adapb <= 1. default is 0.05
...	Further arguments passed to or from other methods.

Value

pc	Crossover rate
pm	Mutation rate

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

fixpcmut, ilmdhc, adana1, adana2, leitingzhi

---

atc *Asymmetric Two-Point Crossover (ATC)*

Description

The Asymmetric Two-Point Crossover (ATC) operator relies on the two-point crossover being implemented differently for Parent1 and Parent2 (Yuan, 2002). Offspring2 is generated by a standard two-point crossover. However, in the generation of Offspring1, the part between the cut points is taken from Parent2, while the other parts are completed from Parent1.

Usage

```
atc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Yuan B. (2002). Deterministic crowding, recombination and self-similarity. In *Proc. of the 2002 Cong. on Evolutionary Computation* (Cat. No. 02TH8600) (Vol. 2, pp. 1516-1521). IEEE.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
atc(parent1, parent2)
```

---

ax *Avarage Crossover*

Description

The AX operator calculates the simple arithmetic mean of the parental chromosomes. Therefore, it is a single-output operator and generates a single offspring (Gwiazda, 2006).

Usage

```
ax(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Gwiazda T.D. (2006). *Genetic Algorithms Reference*. Vol. I: Crossover for Single-Objective Numerical Optimization Problems. Tomaszgwiadze E-books, Poland.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
ax(parent1, parent2, cxon=1)
```

---

bestsol *Best solution monitoring function*

Description

Display best solution from result of GA

Usage

```
bestsol(garesult)
```

Arguments

garesult	GA result object

Value

Display chromosome, fitness value and generation number of best solution.

Author(s)

Zeynel Cebeci & Erkut Tekeli

---

bin2gray *Convert from binary to gray code integer*

Description

The function bin2gray converts a binary coded number to gray coded integer.

Usage

```
bin2gray(bin)
```

Arguments

bin	A binary coded number.

Details

The bin2gray function works as a compliment of the gray2bin function.

Value

Returns the gray coded integer equivalent of the input number.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

gray2bin

Examples

```
bin = c(1,0,1,1)
bin2gray(bin)     # returns 1110

bin = c(1,0,1,0)
bin2gray(bin)     # returns 1111
```

---

bin2int *Convert Binary Numbers to Integers*

Description

The function bin2int converts a binary coded number to integer.

Usage

```
bin2int(bin)
```

Arguments

bin	A binary coded number.

Details

The bin2int function works as a compliment of the int2bin function.

Value

Returns the integer equivalent of the input number.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

int2bin

Examples

```
x <- c(1,1,1,1,1,0,1,0,0)
bin2int(x)  # returns 500
```

---

bitmut *Bit Flip Mutation*

Description

The Bit Flip Mutation operator converts the bit at a randomly selected point to its allele.

This operator is used on binary encoded chromosomes.

Usage

```
bitmut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(1,1,0,1,0,1,0,1,0,0)
bitmut(offspring)
```

`blxa` *Blended Crossover (BLX-α)*

Description

Eshelman and Schaffer (1993) proposed an algorithm called Blended-α Crossover (BLX-α) by introducing the concept of interval scheme to be applied in real-valued problems (Takahashi & Kita, 2001).

Usage

```
blxa(x1, x2, cxon, ...)
```

Arguments

`x1`	A vector. It contains the chromosomal information of parent-1.
`x2`	A vector. It contains the chromosomal information of parent-2.
`cxon`	Number of offspring to be generated as a result of crossover
`...`	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Eshelman, L.J. and Schaffer, J.D. (1993). Real-coded genetic algorithms and interval schemata. In *Foundations of Genetic Algorithms*, Vol. 2, pp. 187-202, Elsevier.

Takahashi, M. and Kita, H. (2001). A crossover operator using independent component analysis for real-coded genetic algorithms. In *Proc. of the 2001 Cong. on Evolutionary Computation (IEEE Cat.No. 01TH8546)*, Vol. 1, pp. 643-649. IEEE.

See Also

`cross`, `px1`, `kpx`, `sc`, `rsc`, `hux`, `ux`, `ux2`, `mx`, `rrc`, `disc`, `atc`, `cpc`, `eclc`, `raoc`, `dc`, `ax`, `hc`, `sax`, `wax`, `lax`, `bx`, `ebx`, `blxab`, `lapx`, `elx`, `geomx`, `spherex`, `pmx`, `mpmx`, `upmx`, `ox`, `ox2`, `mpx`, `erx`, `pbx`, `pbx2`, `cx`, `icx`, `smc`

Examples

```
ebeveyn1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
ebeveyn2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
blxa(ebeveyn1, ebeveyn2, cxon=3)
```

blxab *Blended crossover-$\alpha\beta$ (BLX-$\alpha\beta$)*

Description

Blended crossover-$\alpha\beta$ is another version of the Blended crossover-α operator.

Usage

```
blxab(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
blxab(parent1, parent2)
```

boundmut *Boundary Mutation*

Description

The Boundary Mutation operator is a mutation operator that changes the value of a randomly selected gene in the chromosome with the upper or lower limit value for that gene.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
boundmut(y, lb, ub, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
boundmut(offspring, lb=lb, ub=ub)
```

bsearchmut1 *Boundary Search Mutation 1*

Description

Boundary Search Mutation-1 is an algorithm based on probing the boundaries of the convenience region in constraint processing for NLP optimization (Michalewicz & Schoenauer, 1996). Two genes are randomly selected from the chromosome and one of them is multiplied by a random factor at the q value, while the other gene is multiplied by $1/q$.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
bsearchmut1(y, mutq, ...)
```

Arguments

y	A vector. Chromosome of the offspring
mutq	A number. Value of q
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	A vector. The numbers of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1), 1-32.

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(8, 6, 4, 1, 3)
#set.seed(12)
bsearchmut1(offspring)
mutq = 0.5
#set.seed(12)
bsearchmut1(offspring, mutq=mutq)
```

`bsearchmut2` *Boundary Search Mutation 2*

Description

Boundary Search Mutation-2 is an algorithm based on searching the convenience region boundaries in constraint processing for NLP optimization (Michalewicz & Schoenauer, 1996). Two genes are randomly selected from the chromosome and one is multiplied by the random value p, while the other gene is multiplied by the q value calculated using p.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
bsearchmut2(y, ...)
```

Arguments

`y`	A vector. Chromosome of the offspring
`...`	Further arguments passed to or from other methods.

Value

`mutant`	A vector. Chromosome of the offspring
`mutgen`	A vector. The numbers of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1), 1-32.

See Also

`mutate`, `bitmut`, `randmut`, `randmut2`, `randmut3`, `randmut4`, `unimut`, `boundmut`, `nunimut`, `nunimut2`, `powmut`, `powmut2`, `gaussmut`, `gaussmut2`, `gaussmut3`, `bsearchmut1`, `swapmut`, `invmut`, `shufmut`, `insmut`, `dismut`, `invswapmut`, `insswapmut`, `invdismut`

Examples

```
offspring = c(8, 6, 4, 1, 3)
bsearchmut2(offspring)
```

---

bx *Box Crossover / Flat Crossover*

Description

In the parent chromosomes, the randomly selected value between the minimum and maximum values of each gene is assigned as the value of that gene in the offspring chromosome (Herrera et.al, 1998).

Usage

```
bx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Herrera, F., Lozano, M. and Verdegay J.L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4), 265-319.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
bx(parent1, parent2)
```

`calcM` *Calculate the number of bits in the binary representation of the integer vector*

Description

The function CalcM calculates the number of bits in the binary representation of the integer vector

Usage

```
calcM(ub, ...)
```

Arguments

ub	A vector containing upper bounds
...	Further arguments passed to or from other methods.

Details

This function uses the upper bounds of the integer vector to calculate the number of bits in the binary representation of an integer vector.

Value

A vector of the numbers of bits for binary representation of an integer vector.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`encode4int`

Examples

```
ub = c(10, 10, 10)
calcM(ub)
```

cpc *Count-preserving Crossover (CPC)*

Description

Count-preserving Crossover (CPC) is an operator that assumes the same number of chromosomes equal to 1 in each chromosome in the initial population and tries to preserve this number (Hartley & Konstam, 1993; Gwiazda 2006).

Usage

```
cpc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Hartley S.J. and Konstam A.H. (1993). Using genetic algorithms to generates Steiner triple systems. In *Proc. of the 1993 ACM Conf. on Computer Science* (pp. 366-371).

Gwiazda T.D. (2006). *Genetic Algorithms Reference*. Vol. I: Crossover for Single-Objective Numerical Optimization Problems. Tomaszgwiadze E-books, Poland.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
cpc(parent1, parent2)
```

`cross` *Crossover*

Description

It is a wrapper function that calls crossover operators from a single function.

Usage

```
cross(crossfunc, matpool, cxon, cxpc, gatype, ...)
```

Arguments

`crossfunc`	The name of the crossover operator
`matpool`	A matrix. Mating pool containing selected individuals.
`cxon`	Number of offspring to be generated as a result of crossover
`cxpc`	Crossover Ratio. Default value is 0.95
`gatype`	Indicates the GA type. "gga" is assigned for generational refresh, and "ssga" for steady-state refresh.
`...`	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
genpop = initbin(12,8)                              #Initial population
m = ncol(genpop)-2                                  #Number of Gene
sumx = function(x, ...) (sum(x))                    #Fitness Function
fitvals = evaluate(fitfunc=sumx, genpop[,1:m])      #Fitness Values
genpop[,"fitval"] = fitvals
selidx = select(selfunc=selrws, fitvals)            #Selection of Parents
matpool = genpop[selidx,]                           #Mating Pool
offsprings = cross(crossfunc=px1, matpool=matpool, #Crossing
 cxon=2, cxpc=0.8, gatype="gga")
offsprings
offsprings = cross(crossfunc=kpx, matpool=matpool,
  cxon=2, cxpc=0.8, gatype="ssga", cxps=0.5, cxk=2)
offsprings
```

---

cx *Cycle Crossover (CX)*

Description

The Cycle Crossover (CX) is an algorithm that considers the gene order in the parental chromosomes (Oliver et.al., 1987).

Usage

```
cx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Oliver, I.M., Smith, D. and Holland J.R. (1987). Study of the permutation crossover operators on the traveler salesman problem. In *Grefenstette, J.J. (ed). Genetic Algorithms and Their Applications, Proc. of the 2nd Int. Conf.* Hillsdale, New Jersey: Lawrence Erlbaum, pp. 224-230.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, icx, smc

Examples

```
parent1 =c(9, 8, 2, 1, 7, 4, 5, 0, 6, 3)
parent2 =c(1, 2, 3, 4, 5, 6, 7, 8, 9, 0)
cx(parent1, parent2)
```

---

dc *Discrete Crossover*

Description

The Discrete Crossover (DC) operator is an operator that swaps parent genes if a randomly selected value in the range [0,1] for each gene in the chromosome is greater than or equal to a given threshold value, and does not change if it is less than the threshold value.

Usage

```
dc(x1, x2, cxon, cxps, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxps	Threshold value. 0 <= cxps <= 1
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
dc(parent1, parent2, cxps=0.6)
```

---

decode *Convert from binary number to real number*

Description

The function decode converts a binary number with m digits to a real number between the lower and upper bound.

Usage

```
decode(bin, lb, ub, m)
```

Arguments

bin	A binary number
lb	Lower bound of real number
ub	Upper bound of real number
m	Number of the digits of output value.

Details

This function converts a binary number with m digits to its real equivalent expressed in the range [lb, ub].

Value

Returns the real equivalent of the input number.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

encode

Examples

```
x = c(0,1,0,0,0,0,1,1)
decode(x, lb=50, ub=250, m=8)
```

`decode4int` *Convert binary vectors to integer vectors*

Description

The function decode4int converts each element in a given binary vector to a integer number.

Usage

```
decode4int(x, M, ...)
```

Arguments

`x`	A vector containing binary numbers
`M`	A vector containing the number of bits of each binary in the x.
`...`	Further arguments passed to or from other methods.

Details

This function converts each element in the binary vector passed with the x argument to an integer. The M argument refers to the number of bits of each binary in the x vector.

Value

A vector of integer for input binary vector

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`encode4int`

Examples

```
binmat = c(0,0,1,1,1,0,0,1,0,0,1,0)
M = c(4,4,4)
intmat = decode4int(binmat, M=M)
intmat
```

decodepop *Convert binary number matrix to real number matrix*

Description

The decodepop function generates a real-valued population from a population encoded with binary representation.

Usage

```
decodepop(x, lb, ub, m,  ...)
```

Arguments

x	A vector containing binary numbers
lb	A vector containing lower bounds for variables
ub	A vector containing upper bounds for variables
m	Length for each variable
...	Further arguments passed to or from other methods.

Value

A real-valued matrix

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

encodepop

Examples

```
lb = c(2.5, -2, 0)
ub = c(4.3, 2, 1.5)
eps = c(0.1, 1, 0.01)
#d = nchar(sub('^+','',sub("\.",'',eps)))-1

d = grep('.', strsplit(as.character(eps), '')[[1]])-1
x = round(runif(5, lb[1],ub[1]),d[1])
y = round(runif(5, lb[2],ub[2]),d[2])
w = round(runif(5, lb[3],ub[3]),d[3])
pop = cbind(x, y, w)
pop
encpop = encodepop(pop, lb=lb, ub=ub, eps=eps)
pop = encpop$binmat
m = encpop$m
```

```
decpop = decodepop(pop, lb=lb, ub=ub, m=m)
decpop
for(j in 1:ncol(decpop)) decpop[,j]=round(decpop[,j], d[j])
decpop
```

`disc` *Disrespectful Crossover (DISC)*

Description

Disrespectful Crossover (DISC) is an operator that breaks down similarities or reinforces differences in parental chromosomes (Watson & Pollack, 2000).

Usage

```
disc(x1, x2, cxon, ...)
```

Arguments

`x1`	A vector. It contains the chromosomal information of parent-1.
`x2`	A vector. It contains the chromosomal information of parent-2.
`cxon`	Number of offspring to be generated as a result of crossover
`...`	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Watson R.A. and Pollack J.B. (2000). Recombination without respect: Schema combination and disruption in genetic algorithm crossover. In *Proc. of the 2nd Annual Conf. on Genetic and Evolutionary Computation* (pp. 112-119).

See Also

`cross`, `px1`, `kpx`, `sc`, `rsc`, `hux`, `ux`, `ux2`, `mx`, `rrc`, `atc`, `cpc`, `eclc`, `raoc`, `dc`, `ax`, `hc`, `sax`, `wax`, `lax`, `bx`, `ebx`, `blxa`, `blxab`, `lapx`, `elx`, `geomx`, `spherex`, `pmx`, `mpmx`, `upmx`, `ox`, `ox2`, `mpx`, `erx`, `pbx`, `pbx2`, `cx`, `icx`, `smc`

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
disc(parent1, parent2)
```

dismut *Displacement mutation*

Description

The Displacement mutation cuts the genes between two randomly determined cut-points from the chromosome as a subset and then inserts them, starting from a randomly selected location (Michalewicz, 1992).

This operator is used in problems with permutation encoding.

Usage

```
dismut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutrange	A vector. The numbers of begining and ending of the mutated genes.
r	The number of insertation location.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin-Heidelberg: Springer Verlag.

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
dismut(offspring)
```

ebx *Extended Box Crossover*

Description

Extended Box Crossover (EBX) was proposed by Yoon and Kim (2012) as the more advanced form of Box Crossover (BX). In the EBX operator, the minimum and maximum values are weighted by an alpha factor.

Usage

```
ebx(x1, x2, lb, ub, cxon, cxalfa, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
lb	A vector. Lower bounds of each gene in the chromosomes.
ub	A vector. Upper bounds of each gene in the chromosomes.
cxon	Number of offspring to be generated as a result of crossover
cxalfa	A vector. Alpha value for each gene in the chromosomes. If no value is entered, they are randomly selected by the function in the range [0,1].
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Yoon, Y. and Kim, Y.H. (2012). The roles of crossover and mutation in real-coded genetic algorithms. In *Bioinspired Computational Algorithms anf Their Applications (ed. S. Gao)*, London: INTECH Open Acces Publisher. pp. 65-82.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
lb = c(0, 0, 0, 0, 0, 0)
ub = c(2, 3, 1, 2, 4, 3)
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
ebx(parent1, parent2, lb, ub)
```

---

eclc *Exchange/Linkage Crossover (EC,LC)*

Description

Linkage Crossover (LC) is an operator based on the repositioning of a randomly selected fragment from one of the parents, starting from a randomly selected location in the offspring chromosome (Harik & Goldberg, 1997). It is also called Exchange Crossover (EC).

Usage

```
eclc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Harik, G.D. and Goldberg, D.E. (1997). Learning linkage. *Foundation of Genetic Algorithms* Ch. 4, Morgan-Kaufmann. pp. 247-262.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
eclc(parent1, parent2)
```

`elitism` *Elistist Replacement (Elitism) Function*

Description

The reproduction of individuals with the highest fitness is called elitism. The elitism operator copies a certain number of individuals into the new population. Other individuals are selected from among the offspring in proportion to their fitness values.

Usage

```
elitism(parpop, offpop, reps, ...)
```

Arguments

`parpop`	A matrix. Parent population
`offpop`	A matrix. Offspring population
`reps`	Number of elite individuals
`...`	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`grdelall`, `grmuplambda`, `grmuplambda2`, `grmuplambda3`, `grmuplambda4`, `grmuvlambda`, `grrobin`, `ssrmup1`, `ssrmup1`, `ssrgenitor`, `ssrfamtour`, `ssrx`

elx *Extended-Line Crossover (ELX)*

Description

With the Extended-Line Crossover (ELX) operator, offspring are generated on a line determined by the variable values in the parental chromosomes. ELX identifies the possible line from which offspring can be generated.

Usage

```
elx(x1, x2, lb, ub, cxon, cxealfa, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
lb	A vector. Lower bounds of each gene in the chromosomes.
ub	A vector. Upper bounds of each gene in the chromosomes.
cxon	Number of offspring to be generated as a result of crossover
cxealfa	A number. Expansion rate
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
lb = c(0, 0, 0, 0, 0, 0)
ub = c(2, 3, 1, 2, 4, 3)
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
elx(parent1, parent2, lb, ub, cxealfa=1000)
```

encode *Convert from real number to binary number*

Description

The function encode converts a real number to a binary number with m digits between the given lower bound and upper bound.

Usage

```
encode(real, lb, ub, m)
```

Arguments

real	A real number
lb	Lower bound of real number
ub	Upper bound of real number
m	Number of the digits of output value.

Details

This function converts a real number to its binary equivalent expressed in m digits in the range [lb, ub].

Value

Returns the binary equivalent of the input number.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

decode

Examples

```
x = 102.5
encode(x, lb=50, ub=250, m=8)
```

encode4int *Convert integer vectors to binary vectors*

Description

The function encode4int converts each element in a given integer vector to a binary number.

Usage

```
encode4int(x, M, ...)
```

Arguments

x	A vector containing integer numbers
M	A vector containing the number of bits in the binary representation of each integer variable.
...	Further arguments passed to or from other methods.

Details

This function converts each element in the integer vector passed with the x argument to a binary number. The M argument refers to the number of bits in the binary representation of each integer variable.

Value

A vector of binary representation of input vector

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

decode4int, calcM

Examples

```
n = 5
lb = c(0, 0, 0)
ub = c(10, 10, 10)
set.seed(1)
intmat = matrix(round(runif(3*n, lb, ub)), nr=n, nc=3)
colnames(intmat) = paste0("v",1:3)
head(intmat)
M = calcM(ub)
M
binmat = matrix(NA, nrow=n, ncol=sum(M))
```

```
for(i in 1:n)
  binmat[i,] = encode4int(intmat[i,], M=M)
head(binmat)
```

---

encodepop *Binary encoding of real number matrix*

Description

The encodepop function generates a population encoded with binary representation from a real-valued population given as a matrix.

Usage

```
encodepop(x, lb, ub, eps, ...)
```

Arguments

x	A vector containing real numbers
lb	A vector containing lower bounds for variables
ub	A vector containing upper bounds for variables
eps	Sensitivity vector containing desired sensitivity values for each variable
...	Further arguments passed to or from other methods.

Value

binmat	A binary coded matrix as counterpart of real-valued input matrix
m	A vector containing bit length of each variable

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

decodepop

Examples

```
lb = c(2.5, -2, 0)
ub = c(4.3, 2, 1.5)
eps = c(0.1, 1, 0.01)
#d = nchar(sub('^+','',sub("\.",'',eps)))-1
d = grep('.', strsplit(as.character(eps), '')[[1]])-1
x = round(runif(5, lb[1],ub[1]),d[1])
y = round(runif(5, lb[2],ub[2]),d[2])
w = round(runif(5, lb[3],ub[3]),d[3])
```

```
pop = cbind(x, y, w)
pop
encpop = encodepop(pop, lb, ub, eps)
head(encpop$binmat[,1:10])
m = encpop$m
m
```

---

erx *Edge Recombination Crossover (ERX)*

Description

The Edge Recombination Crossover (ERX) operator ignores outbound directions, that is, it evaluates a chromosome with an undirected edge loop (Whitley et.al., 1989). This operator is based on the concept of neighborhood, as the main idea is to prioritize the edges common to both parents when creating offspring.

Usage

```
erx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Whitley, L.D., Starkweather, T. and D'Ann, F. (1989). Scheduling problems and traveling salesman: the genetic edge recombination operator. In *Proc. of ICGA*, Vol. 89, pp. 133-40.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 3, 5, 6, 4, 2, 8, 7)
parent2 = c(1, 4, 2, 3, 6, 5, 7, 8)
erx(parent1, parent2, cxon=2)
```

`evaluate` *Calculate the fitness values of population*

Description

Calculates the fitness value of a population using the fitness function given with the fitfunc argument.

Usage

```
evaluate(fitfunc, population, objective, ...)
```

Arguments

`fitfunc`	Fitness function
`population`	Population matrix
`objective`	"max" or "min"
`...`	Further arguments passed to or from other methods.

Value

A vector of fitness values for each induvidual in population.

Author(s)

Zeynel Cebeci & Erkut Tekeli

Examples

```
population = initbin()
head(population, 5)
m = ncol(population)-2
fitvals = evaluate(maxone, population[,1:m], objective="max")
head(fitvals, 5)
```

findoptima *Finds peaks and valleys on the curve of a function with single variable*

Description

This function finds the peaks and valleys on the curve of user-defined functions with one variable. The function also plots the function curve that can be used to demonstrate the points for local optima and global optimum in a optimization problem.

Usage

```
findoptima(x, type="max", pflag=TRUE)
```

Arguments

x	a vector of variable
type	Either "max" (the default) or "min". The peaks are found when type="max" and the valleys are found when type="max".
pflag	if this is TRUE, the first and last values are also checked.

Details

The findoptima function finds all the peaks and valleys in a given function curve. The points can be colorized with different colors. See the example below.

Value

Returns a vector of indices of the peaks or valleys on the function curve.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci Z. (2021). *R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari*. Ankara:Nobel Akademik Yayincilik

Examples

```
fx <- function(x) -sin(x)-sin(2*x)-cos(3*x) + 3
x <- seq(-2*pi, 2*pi, by=0.001)
curve(fx, x)
cr <- curve(fx, x, lwd=2)
xy <- cbind(cr$x, cr$y)
peaks <- findoptima(cr$y, type = "max")
valleys <- findoptima(cr$y, type = "min")
```

```
## Finds peaks and valleys
peaks <- findoptima(cr$y, type="max")
valleys <-  findoptima(cr$y, type="min")

## Plotting the function curve and local optima and global optimum
points(xy[peaks,], pch=19, cex=1.2, col=2)
points(xy[valleys,], pch=18, cex=1.2, col=4)
gmin <- valleys[which.min(xy[valleys,2])]
gmax <- peaks[which.min(xy[valleys,2])]
points(xy[gmax,1], xy[gmax,2], pch=19, cex=2, col=2)
points(xy[gmin,1], xy[gmin,2], pch=18, cex=2, col=4)
text(xy[gmax,1], xy[gmax,2], labels="Glob.Max", pos=2, cex=0.8, col=1)
text(xy[gmin,1], xy[gmin,2], labels="Glob.Min", pos=2, cex=0.8, col=1)
```

`fixpcmut` *Static crossover and mutation rate*

Description

The function is used when the crossover and mutation rates are not changed throughout the GA run.

Usage

```
fixpcmut(cxpc, mutpm, ...)
```

Arguments

`cxpc`	Crossover rate
`mutpm`	Mutation rate
`...`	Further arguments passed to or from other methods.

Value

`pc`	Crossover rate
`pm`	Mutation rate

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`ilmdhc`, `adana1`, `adana2`, `leitingzhi`, `adana3`

gaussmut *Gauss Mutation*

Description

Gauss Mutation is an operator made by adding randomly selected values from a normal distribution with a mean of 0 and a standard deviation of sigma to a randomly selected gene in the chromosome (Michalewicz, 1995; Back et.al., 1991; Fogel, 1995).

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
gaussmut(y, mutsdy, ...)
```

Arguments

y	A vector. Chromosome of the offspring
mutsdy	A vector. Vector of standard deviations of genes
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, Z. (1995). Genetic algorithms, numerical optimizations and constraints. In *Proc. of the 6th. Int. Conf. on Genetic Algorithms*, pp. 151-158. Morgan Kaufmann.

Back, T., Hoffmeister, F. and Schwefel, H.F. (1991). A survey of elolution strategies. In *Proc. of the 4th. Int. Conf. on Genetic Algorithms (eds. R.K. Belew and L.B. Booker)*, pp. 2-9. Morgan Kaufmann.

Fogel D.B. (1995). Evolutionary computation. Toward a new philosophy of machine intellegence. Piscataway, NJ: IEEE Press.

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
mutsdy = c(1, 1.5, 1.01, 0.4, 1.5, 1.2)
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
gaussmut(offspring)
```

---

gaussmut2 *Gauss Mutation 2*

Description

Gauss Mutation-2 is an operator by adding a randomly selected value from the standard normal distribution to a randomly selected gene in the chromosome.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
gaussmut2(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
gaussmut2(offspring)
```

gaussmut3 *Gauss Mutation 3*

Description

GM is an operator made by adding randomly selected values from a normal distribution with mean and standard deviation of MU and SIGMA, respectively, to a randomly selected gene in the chromosome.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
gaussmut3(y, mutmy, mutsdy, ...)
```

Arguments

y	A vector. Chromosome of the offspring
mutmy	A vector. Vector of means of genes
mutsdy	A vector. Vector of standard deviations of genes
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
mutmy = c(5, 5, 2, 4, 3, 4)
mutsdy = c(1, 1.5, 1.01, 0.4, 1.5, 1.2)
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
gaussmut(offspring, mutmy=mutmy, mutsdy=mutsdy)
```

geomx *Geometric Crossover*

Description

Geometric Crossover is used to search for applicable region boundaries in constraint processing in NLP problems (Michalewicz & Schoenauer, 1996). It generates one offspring per each cross.

Usage

```
geomx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary Algorithms for constrained parameter optimization problems. *Evoltionary Computation*, 4(1), 1-32.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
geomx(parent1, parent2)
```

gray2bin *Convert gray code to binary integer #1*

Description

The function gray2bin converts gray coded integer to a binary coded number.

Usage

```
gray2bin(gray)
```

Arguments

gray	A gray coded number.

Details

The gray2bin function works as a compliment of the bin2gray function.

Value

Returns the binary coded integer equivalent of the input number.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

bin2gray, gray2bin2

Examples

```
gray = c(1,1,1,0)
gray2bin(gray)       # returns 1011
gray = c(1,1,1,1)
gray2bin(gray)       # returns 1010
```

gray2bin2 *Convert gray code to binary integer #2*

Description

The function gray2bin2 converts a gray-coded integer to a binary-coded number. The conversion function is performed according to the algorithm given by Chakraborty and Janikov (2003).

Usage

```
gray2bin2(gray)
```

Arguments

gray	A gray coded number.

Details

To convert gray coded numbers to binary numbers, a conversion function is defined using the algorithm given by Chakraborty and Janikov (2003). This function is a generic function that does not use the xor operator.

Value

Returns the binary coded integer equivalent of the input number.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Chakraborty, U.K., and Janikow C.Z. (2003). An analysis of Gray versus binary encoding in genetic search. Information Sciences, 156 (3-4), 253-269.

See Also

bin2gray, gray2bin

Examples

```
gray = c(1,1,1,0)
gray2bin2(gray)       # returns 1011

gray = c(1,1,1,1)
gray2bin2(gray)       # returns 1010
```

grdelall *Delete-All Replacement*

Description

All members of the current population are deleted, the new population is created entirely from offspring.

Usage

```
grdelall(parpop, offpop)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrmup1, ssrgenitor, ssrfamtour, ssrx

grmuplambda *Mu+Lambda replacement function 1*

Description

The Mu+Lamda replacement is based on the selection of the fittest parents and offspring as individuals of the new generation population (Smith et.al., 1999; Jenkins et.al., 2019).

Usage

```
grmuplambda(parpop, offpop, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Smith, A.E. and Vavak F. (1999). Replacement strategies in steady state genetic algorithms: Static environments. *Foundations of Genetic Algorithms*. pp. 499-505.

Jenkins, A., Gupta, V., Myrick, A. and Lenoir, M. (2019). Variations of Genetic Algorithms. *arXiv preprint arXiv:1911.00490*.

See Also

grdelall, elitism, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrmup1, ssrgenitor, ssrfamtour, ssrx

---

grmuplambda2 *Mu+Lambda replacement function 2 (delete the worst λ)*

Description

Parents and offspring are ranked separately according to their compatibility among themselves. Then λ offspring with the best fitness value is replaced by λ parent with the worst fitness value.

Usage

```
grmuplambda2(parpop, offpop, lambda, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
lambda	Number of individuals renewed in the population
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrmup1, ssrgenitor, ssrfamtour, ssrx

---

`grmuplambda3` *Mu+Lambda replacement function 3*

Description

After the offspring are ranked according to their fitness values, the λ best fit offspring are replaced by λ parents randomly selected from the current parent population.

Usage

```
grmuplambda3(parpop, offpop, lambda, ...)
```

Arguments

`parpop`	A matrix. Parent population
`offpop`	A matrix. Offspring population
`lambda`	Number of individuals renewed in the population
`...`	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrmup1, ssrgenitor, ssrfamtour, ssrx

grmuplambda4 *Mu+Lambda replacement function 4*

Description

In the current population, randomly selected λ parents are replaced by randomly selected λ offspring.

Usage

```
grmuplambda4(parpop, offpop, lambda, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
lambda	Number of individuals renewed in the population
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuvlambda, grrobin, ssrmup1, ssrgenitor, ssrfamtour, ssrx

grmuvlambda *Mu & Lambda Replacement Function*

Description

In this renewal strategy, after the offspring are ranked according to their fitness values, the number of the population of the offspring with the best fitness value is replaced by the parents (Schwefel, 1981). To use this renewal algorithm, it is necessary to produce many more offspring than the population count.

Usage

```
grmuvlambda(parpop, offpop, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grrobin, ssrmup1, ssrgenitor, ssrfamtour, ssrx

---

grrobin *Round Robin Replacement Function*

Description

The parent and offspring populations are combined. Then, each individual in the combined population is compared with k randomly selected individuals. In these double tournaments, if an individual has higher fitness than the individual they are compared to, +1 point is obtained. The new population is created from the individuals with the highest score.

Usage

```
grrobin(parpop, offpop, repk, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
repk	Number of Comparisons
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, ssrmup1, ssrgenitor, ssrfamtour, ssrx

---

hc *Heuristic Crossover*

Description

The Heuristic Crossover (HC) operator is a conditional operator (Herrera et.al, 1998; Umbarkar & Sheth, 2005). A random r value is generated in the range [0,1]. Then if Parent2's fitness value is greater than or equal to Parent1's fitness value, the difference between them is weighted by r and added to Parent2. It is subtracted in minimization problems. This operator produces a single offspring, but due to the random value of r, repeated offspring may result in different offspring.

Usage

```
hc(x1, x2, fitfunc, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
fitfunc	Fitness Function
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Herrera, F., Lozano, M. and Verdegay, J.L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intellegence Review*, 12(4), 265-319 Umbarkar, A.J. and Sheth P.D. (2015). Crossover operators in genetic algorithms: A riview, *ICTACT Journal on Soft Computing*, 6(1), 1083-1092.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
fitfunc = function(x, ...) 2*(x[1]-1)^2 + 5*(x[2]-2)^2 + 10
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
hc(parent1, parent2, fitfunc)
```

---

hgaoptim *GA + optim hybridization function*

Description

This function allows GA to hybridize with methods in the optim general-purpose optimization function for n-variable problems in R's basic *stats* package (R Core Team, 2021).

Usage

```
hgaoptim(genpop, fitfunc, hgaparams,
                    hgaftype, hgans, ...)
```

Arguments

genpop	A matrix of individuals in the current population and their fitness values.
fitfunc	Fitness function
hgaparams	A list of parameters defined for use by the Optim function.
hgaftype	Types of fitness to transfer. • w: individuals with the worst fitness value • b: individuals with the best fitness value • r: randomly selected individuals
hgans	Number of individuals to be transferred to the Optim.
...	Further arguments passed to or from other methods.

Value

A matrix containing the updated population.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

R Core Team. (2021). R: A language and environmental for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

See Also

hgaoptimx, hgaroi

Examples

```
hgaparams = list(method="Nelder-Mead", poptim=0.05, pressel=0.5,
  control = list(fnscale=1, maxit=100))
n = 5                   # Size of population
m = 2                   # Number of variables
lb = c(-5.12, -5.12)    # Lower bounds for sample data
ub = c(5.12, 5.12)      # Upper bounds for sample data
genpop = initval(n, m, lb=lb, ub=ub) # Sample population
fitfunc = function(x, ...) 2*(x[1]-1)^2 + 5*(x[2]-2)^2 + 10
fitvals = evaluate(fitfunc, genpop[,1:m])
genpop[,"fitval"]=fitvals
genpop
newpop = hgaoptim(genpop, fitfunc, hgaparams, hgaftype="r", hgans=3)
newpop
```

---

hgaoptimx *GA + optimx hybridization function*

Description

This function allows GA to hybridize with methods in the *optimx* package (Nash & Varadhan, 2011; Nash, 2014).

Usage

```
hgaoptimx(genpop, fitfunc, hgaparams,
                    hgaftype, hgans, ...)
```

Arguments

genpop	A matrix of individuals in the current population and their fitness values.
fitfunc	Fitness function
hgaparams	A list of parameters defined for use by the Optim function.
hgaftype	Types of fitness to transfer. • w: individuals with the worst fitness value • b: individuals with the best fitness value • r: randomly selected individuals
hgans	Number of individuals to be transferred to the Optim.
...	Further arguments passed to or from other methods.

Value

A matrix containing the updated population.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Nash, J.C. and Varadhan, R. (2011). Unified optimization algorithms to aid software system users: optimx for R. *Journal of Statistical Software*, 43(9), 1-14. URL http://www.jstatsoft.org/v43/i09. Nash, J.C. (2014). On best practice optimization methods in R. *Journal of Statistical Software*, 60(2), 1-14. URL http://www.jstatsoft.org/v60/i02.

See Also

hgaoptim, hgaroi

Examples

```
n = 5                                # Size of population
m = 2                                # Number of Variables
lb = c(-5.12, -5.12)                 # Lower bounds of sample data
ub = c(5.12, 5.12)                   # Upper bounds of sample data
hgaparams = list(method="L-BFGS-B",
  poptim=0.05, pressel=0.5,
  lower=lb, upper=ub,
  control=list(maximize=FALSE, maxit=100))
genpop = initval(n, m, lb=lb, ub=ub) # Sample population
fitfunc = function(x, ...) 2*(x[1]-1)^2 + 5*(x[2]-2)^2 + 10
fitvals = evaluate(fitfunc, genpop[,1:m])
genpop[,"fitval"]=fitvals
genpop
genpop = hgaoptimx(genpop, fitfunc, hgaparams, hgaftype="r", hgans=3)
genpop
```

---

hgaroi *GA + ROI hybridization function*

Description

This function allows GA to hybridize with methods in the *ROI* package (Theussl et.al., 2020).

Usage

```
hgaroi(genpop, fitfunc, hgaparams,
                  hgaftype, hgans, ...)
```

Arguments

genpop	A matrix of individuals in the current population and their fitness values.
fitfunc	Fitness function
hgaparams	A list of parameters defined for use by the Optim function.
hgaftype	Types of fitness to transfer. • w: individuals with the worst fitness value • b: individuals with the best fitness value • r: randomly selected individuals
hgans	Number of individuals to be transferred to the Optim.
...	Further arguments passed to or from other methods.

Value

A matrix containing the updated population.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Theussl, S., Schwendinger, F. and Hornik, K. (2020). ROI: An extensible R optimization infrastructure. *Journal of Statistical Software*, 94(15), 1-64.

See Also

hgaoptim, hgaoptimx

Examples

```
n = 5                                 # Size of population
m = 2                                 # Number of variable
lb = c(-5.12, -5.12)                  # Lower bounds of sample data
ub = c(5.12, 5.12)                    # Upper bounds of sample data
hgaparams = list(method="L-BFGS-B",
  poptim=0.05, pressel=0.5,
  lower=lb, upper=ub,
  control=list(maxit=100))
genpop = initval(n, m, lb=lb, ub=ub)  # Sample population
fitfunc = function(x, ...) 2*(x[1]-1)^2 + 5*(x[2]-2)^2 + 10
fitvals = evaluate(fitfunc, genpop[,1:m])
genpop[,"fitval"]=fitvals
genpop
genpop = hgaroi(genpop, fitfunc, hgaparams,
  hgaftype="r", hgans=3)
genpop
```

hux *Heuristic Uniform Crossover*

Description

"Heuristic Uniform Crossover" is an algorithm that works by detecting genes that differ to control the level of disruption in Parental chromosomes (De Jong & Spears, 1991).

Usage

```
hux(x1, x2, cxon, cxps, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxps	It determines the rate of gene exchange between the chromosomes of the parents.
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

De Jong, K.A. and Spears, W. (1991). On the virtues of parameterized uniform crossover. In *Proc. of the 4th Int. Conf. on Genetic Algorithms*. Morgan Kaufman Publishers.

See Also

cross, px1, kpx, sc, rsc, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
hux(parent1, parent2)
```

icx *Improved Cycle Crossover (ICX)*

Description

ICX is based on a deterministic algorithm that can produce up to 2 offspring (Hussain et.al., 2018).

Usage

```
icx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Hussain, A., Muhammad, Y.S. and Sajid, M.N. (2018). An improved genetic algorithm crossover operator for traveling salesman problem. *Turkish Journal of Mathematics and Computer Science*, 9, 1-13.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, smc

Examples

```
parent1 = c(3, 4, 8, 2, 7, 1, 6, 5)
parent2 = c(4, 2, 5, 1, 6, 8, 3, 7)
icx(parent1, parent2)
```

`ilmdhc` *ILM/DHC adaptation function*

Description

ILM/DHC is an adaptive function with an increasing low mutation rate (ILM) and a decreasing high crossover rate (DHC) as the generation progresses (Hassanat et.al., 2019).

Usage

```
ilmdhc(g, gmax, ...)
```

Arguments

`g`	Current generation
`gmax`	Maximum generation
`...`	Further arguments passed to or from other methods.

Value

`pc`	Crossover rate
`pm`	Mutation rate

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A. and Prasath, V.B. (2019). Choosing mutation and crossover ratios for genetic algorithm: A review with a new dynamic approach. *Information*, 10(12), 390.

See Also

`fixpcmut`, `adana1`, `adana2`, `leitingzhi`, `adana3`

Examples

```
N = 50
gmax = 1000
g = c(1, 10, 50, 100, 250, 500, 750, gmax)
pc = ilmdhc(g=g, gmax=gmax)$pc
pc
nc = round(pc*N)
nc
pm = ilmdhc(g=g, gmax=gmax)$pm
pm
```

```
nm = round(pm*N)
nm
nm = ifelse (!nm, 1, nm)
nm
plot(pm, type="l", col=4, lwd=2, lty=1, xaxt="n", ylab="Ratio", xlab="Generation")
lines(pc, type="l", col=2, lwd=2, lty=2)
legend("top", inset=.02, c("pm","pc"), col=c(4,2), lty=c(1,2), horiz=TRUE, cex=0.8)
axis(1, at=1:length(g),labels=g, col.axis="red", las=2)
```

`initbin` *Initialize the population with binary encoding*

Description

The initbin function is an initialization function that can be used for binary encoding. It generates an initial population of population size n and chromosome length m.

Usage

```
initbin(n, m, prevpop, type, ...)
```

Arguments

`n`	Population size
`m`	Chromosome length
`prevpop`	Matrix of solutions used in heuristic and hybrid initialization
`type`	Type of output matrix
`...`	Further arguments passed to or from other methods.

Value

The output matrix includes only chromosomes of initial population when `type=2`, otherwise The output matrix includes chromosomes of initial population and additional two empty columns for generation number and fitness values.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`initval`, `initperm`, `initnorm`, `initialize`

Examples

```
n = 20  #Population size (number of chromosemes)
m = 5   #Number of gene (chromosome length)
population = initbin(n, m)
head(population, 4)
tail(population, 4)
```

`initialize` *Initialize function*

Description

The initialize function is a function that wraps various initialization functions.

Usage

```
initialize(initfunc, n, m, type, ...)
```

Arguments

`initfunc`	Initialization function
`n`	Population size
`m`	Chromosome length (number of variables)
`type`	Type of output matrix
`...`	Further arguments passed to or from other methods.

Value

The output matrix includes only chromosomes of initial population when `type=2`, otherwise The output matrix includes chromosomes of initial population and additional two empty columns for generation number and fitness values.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`initbin`, `initval`, `initperm`, `initnorm`

Examples

```
initpop = initialize(initfunc=initbin, n=6, m=4)
initpop
```

`initnorm` *Normal distribution based initialization*

Description

The pmean and psd arguments of this function represent the mean and standard deviation of a normally distributed population, respectively. Using these parameters, the function generates a random initial population with n individuals and m variables.~

Usage

```
initnorm(n, m, pmean, psd, type, ...)
```

Arguments

`n`	Population size
`m`	Chromosome length (number of variables)
`pmean`	Mean of normal distribution
`psd`	Standard deviation of normal distribution
`type`	Type of output matrix
`...`	Further arguments passed to or from other methods.

Value

The output matrix includes only chromosomes of initial population when `type=2`, otherwise The output matrix includes chromosomes of initial population and additional two empty columns for generation number and fitness values.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`initbin`, `initval`, `initperm`, `initialize`

Examples

```
initpop = initialize(initfunc=initnorm, n=20, m=5,
  pmean=50, psd=5, type=2)
head(initpop,3)
```

`initperm` *Permutation coded initialization*

Description

This function generates an initial population when each variable of the chromosomes is desired to be encoded on a rank scale or permutation.

Usage

```
initperm(n, permset, prevpop, type, ...)
```

Arguments

`n`	Population size
`permset`	A vector of permutation set
`prevpop`	Matrix of solutions used in heuristic and hybrid initialization
`type`	Type of output matrix
`...`	Further arguments passed to or from other methods.

Details

Unlike other initialization function inputs, the initperm function has an argument called permset. This argument is a vector containing permutation set values. The permutation set can contain numbers or letters. In the initial population, each variable randomly takes any value from this set, but there cannot be two of the same value in a chromosome.

Value

The output matrix includes only chromosomes of initial population when `type=2`, otherwise The output matrix includes chromosomes of initial population and additional two empty columns for generation number and fitness values.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`initbin`, `initval`, `initnorm`, `initialize`

Examples

```
n = 20   #Population size (number of chromosemes)
m = 6    #number of Variables
lb = c(10, 2, 5, 100, 50, 25)
ub = c(40, 8, 20, 500, 250, 90)
population = initval(n, m, lb=lb, ub=ub, nmode="integer")
tail(population, 3)
```

`initval` *Value encoded initialization*

Description

Initialize the population with integer or real numbers

Usage

```
initval(n, m, prevpop, lb, ub, nmode="real", type, ...)
```

Arguments

`n`	Population size
`m`	Chromosome length (number of variables)
`prevpop`	Matrix of solutions used in heuristic and hybrid initialization
`lb`	Lower bound of each variables
`ub`	Upper bound of each variables
`nmode`	Type of variables ("integer" or "real")
`type`	Type of output matrix
`...`	Further arguments passed to or from other methods.

Details

With this function, populations are initialized with integer and/or real numbers depending on the GA problem. In this case, the value type must be known. Furthermore, the lower and upper bound values for each variable must be known. If desired, heuristic or mixed initialization can be done with the `prevpop` argument.

Value

The output matrix includes only chromosomes of initial population when `type=2`, otherwise The output matrix includes chromosomes of initial population and additional two empty columns for generation number and fitness values.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

initbin, initperm, initnorm, initialize

Examples

```
n = 15  #Population size (number of chromosemes)
m = 4   #number of Variables
population = initval(n, m)
head(population, 3)
tail(population, 3)
```

---

insmut *Insertation Mutation*

Description

SM is inserted back in a different place into the chromosome by removing a randomly selected gene from the chromosome.

This operator is used in problems with permutation encoding.

Usage

```
insmut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.
mutpoint	The number of inserted location of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
insmut(offspring)
```

`insswapmut` *Insertion + Inversion Mutation*

Description

It is a mutation operator that combines insertion and inversion mutation.

This operator is used in problems with permutation encoding.

Usage

```
insswapmut(y, ...)
```

Arguments

`y`	A vector. Chromosome of the offspring
`...`	Further arguments passed to or from other methods.

Value

`mutant`	A vector. Chromosome of the offspring
`mutgen`	A vector. The numbers of begining and ending of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`mutate`, `bitmut`, `randmut`, `randmut2`, `randmut3`, `randmut4`, `unimut`, `boundmut`, `nunimut`, `nunimut2`, `powmut`, `powmut2`, `gaussmut`, `gaussmut2`, `gaussmut3`, `bsearchmut1`, `bsearchmut2`, `swapmut`, `invmut`, `shufmut`, `insmut`, `dismut`, `invswapmut`, `invdismut`

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
insswapmut(offspring)
```

int2bin *Convert an integer to binary coded number*

Description

The function int2bin converts integers to binary coded numbers.

Usage

```
int2bin(int, m)
```

Arguments

int	Input number (integer)
m	Number of the digits of output value.

Value

Returns the binary coded number of the integer number given in the input.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

int2bin

Examples

```
int2bin(250)      # returns 11111010
int2bin(250, 9)   # returns 011111010
```

invdismut *Displacement + Inversion Mutation*

Description

It is a mutation operator that combines displacement and inversion mutation.

This operator is used in problems with permutation encoding.

Usage

```
invdismut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutrange	A vector. The numbers of begining and ending of the mutated genes.
r	The number of insertation location.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
invdismut(offspring)
```

---

invmut *Inversion Mutation*

Description

Inversion Mutation selects a subset of genes and inverses the genes in the subset (Hollad, 1975; Fogel, 1990).

This operator is used in problems with permutation or binary encoding.

Usage

```
invmut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutrange	A vector. The numbers of begining and ending of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Holland, J. (1975). *Adaptation in Naturel and Articial Systems*, Ann Arbor: University of Michigan Press.

Fogel D.B. (1995). Evolutionary computation. Toward a new philosophy of machine intellegence. Piscataway, NJ: IEEE Press.

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
invmut(offspring)
```

---

invswapmut *Swap + Inversion Mutation*

Description

It is a mutation operator that combines swap and inversion mutation.

This operator is used in problems with permutation encoding.

Usage

```
invswapmut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	A vector. The numbers of begining and ending of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, insswapmut, invdismut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
invswapmut(offspring)
```

---

kpx *k-point Crossover*

Description

In the k-PX cross, the parent chromosomes are cut from two or more points and transferred to the offspring, providing more diversity.

Usage

```
kpx(x1, x2, cxon, cxk, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxk	Number of cut points
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
kpx(parent1, parent2)
```

lapx *Laplace Crossover*

Description

Laplace Crossover (LAPX) is a crossover operator that uses a location parameter and a scaling parameter (Krishnamoorthy, 2006; Deep et.al., 2009).

Usage

```
lapx(x1, x2, cxon, cxa, cxb, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxa	Location Parameter
cxb	Scale Parameter. (cxb > 0)
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Krishnamoorthy, K. (2006). *Handbook of Statistical Distributions with Applications*. Chapman & Hall/CRC

Deep, K., Singh, K.P., Kansal, M.L. and Mohan, C. (2009). A real-coded genetic algorithm for solving integer and mixed integer optimization problems. *Applied Mathematics and Computation*, 212(2): 505-518.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
lapx(parent1, parent2, cxon=3)
```

lax *Local Arithmetic Crossover*

Description

New offspring are generated by applying an arithmetic mean on the parents' chromosomes with a different random weight for each gene.

Usage

```
lax(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
lax(parent1, parent2, cxon=3)
```

`leitingzhi` *Lei & Tingzhi Adaptation Function*

Description

This adaptation function proposed by Lei & Tingzhi (1994) is an adaptation function that takes into account the cooperation of individuals.

Usage

```
leitingzhi(fitvals, cxpc, cxpc2,
                      mutpm, mutpm2, adapa, adapb, ...)
```

Arguments

`fitvals`	A vector. Fitness values of current generation
`cxpc`	Crossover rate for adaptation. 0 <= cxpc <= 1. default is 0.9
`cxpc2`	Crossover rate for adaptation. 0 <= cxpc2 <= 1. default is 0.5
`mutpm`	Mutation rate for adaptation. 0 <= mutpm <= 1. default is 0.05
`mutpm2`	Mutation rate for adaptation. 0 <= mutpm2 <= 1. default is 0.2
`adapa`	Adaptation threshold for average of fitness values. 0 <= adapa <= 1. default is 0.7
`adapb`	Adaptation threshold for minimum of fitness values. 0.5 <= adapb <= 1. default is 0.5
`...`	Further arguments passed to or from other methods.

Value

`pc`	Crossover rate
`pm`	Mutation rate

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Lei, W. and Tingzhi, S. (2004). An improved adaptive genetic algorithm and its application to image segmentation. In *Proc. of 5th Int. Conf. on Artificial Neural Network and Genetic Algorithms*, pp. 112-119.

See Also

`fixpcmut`, `ilmdhc`, `adana1`, `adana2`, `adana3`

maxone *MAXONE fitness function*

Description

Fitness function that calculates the number of 1s in each individual

Usage

```
maxone(x, ...)
```

Arguments

x	A vector
...	Further arguments passed to or from other methods.

Value

Number of 1s

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

maxone1, maxone2, minone

Examples

```
C2 = c(1, 1, 1, 0, 1, 0, 0, 0)
maxone(C2)
C3 = c(1, 1, 1, 1, 1, 1, 1, 1)
maxone(C3)
```

maxone1 *MAXONE1 fitness function*

Description

Fitness function that calculates the number of 1s in each individual

Usage

```
maxone1(x, ...)
```

Arguments

x	A vector
...	Further arguments passed to or from other methods.

Value

Number of 1s

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

maxone, maxone2, minone

Examples

```
C2 = c(1, 1, 1, 0, 1, 0, 0, 0)
maxone1(C2)
C3 = c(1, 1, 1, 1, 1, 1, 1, 1)
maxone1(C3)
```

---

maxone2 *maxone2 fitness function*

Description

Calculates the sum of each row of a matrix or data frame.

Usage

```
maxone2(x, ...)
```

Arguments

x	A matrix or a data frame
...	Further arguments passed to or from other methods.

Value

A vector includes sum of each row in a matrix or data frame

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

maxone1, maxone, minone

Examples

```
binmat = matrix(nrow=5, ncol=8, byrow=TRUE, c(
1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 0, 1, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0
))
rownames(binmat) = paste0("C",1:5)
maxone2(binmat)
```

---

minone *minone fitness function*

Description

Calculates the inverse of the sum of each row of a matrix or data frame.

Usage

```
minone(x, ...)
```

Arguments

x	A matrix or a data frame
...	Further arguments passed to or from other methods.

Value

A vector includes the inverse of the sum of each row in a matrix or data frame

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

maxone, maxone1, maxone2

Examples

```
binmat = matrix(nrow=5, ncol=8, byrow=TRUE, c(
1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 0, 1, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0
))
rownames(binmat) = paste0("C",1:5)
minone(binmat)
```

`monprogress` *Monitor Fitness Value Progress*

Description

Monprogress function performs by creating a line plot of the best fitness value found across generations.

Usage

```
monprogress(g, genfits, objective, ...)
```

Arguments

`g`	Generation number
`genfits`	A matrix for fitness values
`objective`	Type of optimization. "min" or "max"
`...`	Further arguments passed to or from other methods.

Value

No return value, called for the side effect of drawing a plot.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

show

Examples

```
n = 100
genfits = matrix(NA, nrow=n, ncol=5)
genfits[1,3] = 50
objective = "max"
for(i in 1:(n-1)){
  g=i
  monprogress(g=g, genfits=genfits, objective=objective)
  genfits[g+1, 3] = genfits[g, 3] + runif(1, -2, 5)
}
```

---

mpmx *Modified Partially Mapped Crossover*

Description

Modified Partially Mapped Crossover (MPMX) is a crossover operator for permutation encoded chromosomes. Each of the offspring uses sequencing information partially determined by each of their parents. Two different cut points are randomly determined. The part outside of the two cut points is replaced. Pieces between the two cut points are complemented from the original parental genes. However, if the same genes are found among the copied genes, they are changed.

Usage

```
mpmx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(0, 8, 4, 5, 6, 7, 1, 2, 3, 9)
parent2 = c(6, 7, 1, 2, 4, 8, 3, 5, 9, 0)
mpmx(parent1, parent2)
```

---

mpx *Maximal Preservative Crossover (MPX)*

Description

The Maximal Preservative Crossover (MPX) operator is an operator that tries to preserve good edges but ensure adequate gene exchange between parents (Muhlenbein et.al., 1988).

Usage

```
mpx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Muhlenbein, H., Gorges-Schleuter, M. and Kramer, O. (1988). Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7(1), 65-85.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(0, 8, 4, 5, 6, 7, 1, 2, 3, 9)
parent2 = c(6, 7, 1, 2, 4, 8, 3, 5, 9, 0)
mpx(parent1, parent2)
```

mutate *Function of Mutation Application*

Description

With mutation, the chromosomes of individuals are randomly changed and sent to the next generation.

Usage

```
mutate(mutfunc, population, mutpm, gatype, ...)
```

Arguments

mutfunc	The name of the mutation operator
population	A matrix. Population of offspring to be mutated
mutpm	Mutation Rate
gatype	Indicates the GA type. "gga" is assigned for generational refresh, and "ssga" for steady-state refresh.
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the mutated offsprings

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offsprings=initbin(25,5)
offsprings[,"fitval"] = evaluate(maxone, offsprings[,1:(ncol(offsprings)-2)])
head(offsprings, 4)          # mutant individual may be further ahead.
mutatedpop = mutate(mutfunc=bitmut, population=offsprings, mutpm=0.1, gatype="gga")
mutatedpop[,"fitval"] = evaluate(maxone, mutatedpop[,1:(ncol(mutatedpop)-2)])
head(mutatedpop, 4)
```

mx *Mask crossover*

Description

This crossover function copies parent1 and parent2 to offspring1 and offspring2, respectively. A vector of length m is then randomly generated for each parent, containing the values 0 and 1. Elements in this vector are then compared for each gene location. If the element at the ith position of the first vector is equal to that of the second vector, no change is made. However, if the first is 0 and the second is 1, the ith gene of Parent2 is copied as the ith gene of Offspring1. If the ith elements of the vectors are 1 and 0, the i th gene of Parent1 is copied as the i th gene of Offspring2 (Louis & Rawlins, 1991).

Usage

```
mx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Louis S.J. and Rawlins G.J. (1991). Designer Genetic Algorithms: Genetic Algorithms in Structure Design. In *4th Int. Conf. on Genetic Algorithms*. (pp. 53-60)

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
mx(parent1, parent2, cxon=3)
```

nunimut *Non-uniform Mutation*

Description

The nunimut operator is a mutation operator that adjusts for generations by reducing the mutation severity according to genetic progression (Michalewicz, 1994).

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
nunimut(y, lb, ub, g, gmax, mutb, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
g	Current generation number.
gmax	Maximum generation number.
mutb	An exponent parameter that sets non-uniformity
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, . (1994).

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
nunimut(offspring, lb=lb, ub=ub, g=1, gmax=100, mutb=0.5)
set.seed(12)
nunimut(offspring, lb=lb, ub=ub, g=50, gmax=100, mutb=0.5)
```

---

nunimut2 *Adaptive Non-uniform mutation*

Description

This operator is an adaptive mutation operator that increases the probability of the mutation severity approaching 0 as the number of generations increases.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
nunimut2(y, lb, ub, g, gmax, mutb, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
g	Current generation number.
gmax	Maximum generation number.
mutb	An exponent parameter that sets non-uniformity
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
nunimut2(offspring, lb=lb, ub=ub, g=1, gmax=100, mutb=0.5)
set.seed(12)
nunimut2(offspring, lb=lb, ub=ub, g=50, gmax=100, mutb=0.5)
```

---

ox *Order Crossover (OX)*

Description

Order Crossover (OX) is a crossover operator for permutation encoded chromosomes. It is a different variant of PMX and it receives a part of the offspring chromosome from Parent1 and the remaining part from Parent2 in a certain sequence (Davis, 1985).

Usage

```
ox(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Davis, L. (1985). Appliying adaptive algorithms to epistatic domains. In *Proc. of the Int. Joint Conf. on Artificial Intellegence*, Vol. 85, pp. 162-164.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(3, 4, 8, 2, 7, 1, 6, 5)
parent2 = c(4, 2, 5, 1, 6, 8, 3, 7)
ox(parent1, parent2)
```

---

ox2 *Order-based crossover (OX2)*

Description

Order-based crossover (OX2) is a crossover operator for permutation encoded chromosomes. It is an operator that forces the order of several randomly selected positions in one parent to the other parent (Syswerda, 1991).

Usage

```
ox2(x1, x2, cxon, cxoxk, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxoxk	Number of genes to be changed
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Sysweda, G. (1991). Schedule optimization using genetic algorithms. *Handbook of Genetic Algorithms*.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 2, 3, 4, 5, 6, 7, 8)
parent2 = c(4, 2, 5, 1, 6, 8, 3, 7)
ox2(parent1, parent2)
```

---

pbx *Position-Based Crossover (PBX)*

Description

The Position-Based Crossover (PBX) operator inserts a different number of randomly selected genes in one parent into the same position in Offspring1, then rounds off the other genes in sequence according to their positions in the other parent (Syswerda, 1991). Other offspring are generated similarly if desired or necessary. PBX is an operator that tries to ensure diversity in recombination while taking care to preserve position.

Usage

```
pbx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Syswerda, G. (1991). Schedule optimization using genetic algorithms. *Handbook of Genetic Algorithms*.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx2, cx, icx, smc

Examples

```
parent1 = c(3, 4, 8, 2, 7, 1, 6, 5)
parent2 = c(4, 2, 5, 1, 6, 8, 3, 7)
pbx(parent1, parent2, cxon=2)
```

---

pbx2 *Position-Based Crossover 2 (PBX2)*

Description

The Position-Based Crossover (PBX) operator inserts a different number of randomly selected genes in one parent into the same position in Offspring1, then rounds off the other genes in sequence according to their positions in the other parent (Syswerda, 1991). Other offspring are generated similarly if desired or necessary. PBX is an operator that tries to ensure diversity in recombination while taking care to preserve position.

Usage

```
pbx2(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Syswerda, G. (1991). Schedule optimization using genetic algorithms. *Handbook of Genetic Algorithms*.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, cx, icx, smc

Examples

```
ebeveyn1 = c(3, 4, 8, 2, 7, 1, 6, 5)
ebeveyn2 = c(4, 2, 5, 1, 6, 8, 3, 7)
pbx2(ebeveyn1, ebeveyn2, cxon=2)
```

---

plotfitness *Fitness statistics graph by GA generations*

Description

Fitness statistics graph by GA generations

Usage

```
plotfitness(genfits, options)
```

Arguments

genfits	A matrix. Best fitness of each generation
options	A vector. Statistics to be plotted.

- 1: minimum
- 2: maximum
- 3: average
- 4: Q1
- 5: median
- 6: Q3

Value

No return value, called for the side effect of drawing a plot.

Author(s)

Zeynel Cebeci & Erkut Tekeli

pmx *Partially Mapped Crossover*

Description

Partially Mapped Crossover (PMX) is the most commonly used crossover operator for permutation encoded chromosomes. Each of the offspring uses sequencing information partially determined by each of their parents (Goldberg & Lingle, 1985). Two different cut points are randomly determined. The part between the two cut points is replaced. Pieces outside of the two cut points are complemented from the original parental genes. However, if the same genes are found among the copied genes, they are changed.

Usage

```
pmx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Goldberg, D.E. and Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. In *Proc. of an international conference on genetic algorithms and their applications*. Vol. 154, pp. 154-159. Carnegie-Mellon University, Pittsburgh, PA.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(3, 4, 8, 2, 7, 1, 6, 5)
parent2 = c(4, 2, 5, 1, 6, 8, 3, 7)
pmx(parent1, parent2, cxon=2)
```

powmut *Power Mutation*

Description

Power Mutation is an operator that generates a mutation in a random gene at a certain power of a random number.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
powmut(y, lb, ub, mutpow, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
mutpow	An exponent parameter
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
powmut(offspring, lb=lb, ub=ub, mutpow=3)
```

powmut2 *Power Mutation 2*

Description

Power Mutation is an operator that generates a mutation in a random gene at a certain power of a random number. In this operator, a different exponent parameter can be given for each gene.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
powmut2(y, lb, ub, mutpow, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
mutpow	A vector of exponent parameter
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
mutpow = c(3, 0.5, 0.5, 2, 3, 1)
offspring = c(8, 6, 4, 1, 3, 7)
set.seed(12)
powmut2(offspring, lb=lb, ub=ub, mutpow=mutpow)
```

px1 *One-point Crossover*

Description

One-point crossover is where randomly selected parent chromosomes from the mating pool are cut at one point and then recombine to generate off-springs.

Usage

```
px1(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
px1(parent1, parent2)
```

randmut *Random Resetting Mutation*

Description

The Random Resetting Mutation operator replaces the value of a randomly selected gene with a randomly selected value between the allowed limits for that gene.

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
randmut(y, lb, ub, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
randmut(offspring, lb, ub)
```

randmut2 *Random mutation 2*

Description

For each gene, if a random number is less than the mutation rate, the gene's value is modified by adding a random value selected from the normal distribution with a mean of zero and a standard deviation of 0.1x(ub-lb).

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
randmut2(y, lb, ub, mutpm, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
mutpm	Mutation rate
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

mutate, bitmut, randmut, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c( 2,  1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
randmut2(offspring, lb=lb, ub=ub, mutpm=0.1)
```

randmut3 *Random mutation 3*

Description

For each gene, if a random number is less than the mutation rate, the gene's mean value is zero and its standard deviation is |ub-lb| The random value selected from the normal distribution is changed by adding it (Yoon & Kim, 2012).

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
randmut3(y, lb, ub, mutpm, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
mutpm	Mutation rate
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Yoon, Y. and Kim, Y.H. (2012). The roles of crossover and mutation in real-coded genetic algorithms. In *Bioinspired Computational Algorithms and Their Applications (ed. S. Gao)*. London: INTECH Open Access Publisher. pp. 65-82.

See Also

mutate, bitmut, randmut, randmut2, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
randmut3(offspring, lb=lb, ub=ub, mutpm=0.1)
```

`randmut4` *Random mutation 4*

Description

An alternative random mutation operator proposed by Wijayaningrum et.al. (2017).

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
randmut4(y, lb, ub,...)
```

Arguments

`y`	A vector. Chromosome of the offspring
`lb`	A vector. Lower bounds of genes
`ub`	A vector. Upper bounds of genes
`...`	Further arguments passed to or from other methods.

Value

`mutant`	A vector. Chromosome of the offspring
`mutgen`	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Wijayaningrum, V.N., Starkweather, T. and D'ann, F. (2017). Scheduling problemsand traveling salesman: The genetic edge recombination operators. In *Proc. of ICGA*, Vol. 89, pp. 133-40.

See Also

`mutate`, `bitmut`, `randmut`, `randmut2`, `randmut3`, `unimut`, `boundmut`, `nunimut`, `nunimut2`, `powmut`, `powmut2`, `gaussmut`, `gaussmut2`, `gaussmut3`, `bsearchmut1`, `bsearchmut2`, `swapmut`, `invmut`, `shufmut`, `insmut`, `dismut`, `invswapmut`, `insswapmut`, `invdismut`

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
randmut4(offspring, lb=lb, ub=ub)
```

---

raoc *Randomized And/Or Crossover (RAOC)*

Description

The Randomized And/Or Crossover (RAOC) operator processes parental chromosomes with AND/OR. According to the value of a randomly selected number, one of the offspring is determined by AND and the other is determined by the OR operation.

Usage

```
raoc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
raoc(parent1, parent2)
```

rrc *Random Respectful Crossover (RRC)*

Description

It is a crossover function that transfers genes that are equal at a particular locus on the parent chromosomes to the offspring as they are while transferring the different ones randomly (Radcliffe, 1991).

Usage

```
rrc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Radcliffe N.J. (1991). Forma analysis and Random Restpectful Recombination. In *4th Int. Conf. on Genetic Algorithms*. Vol. 91, pp. 222-229.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
rrc(parent1, parent2)
```

rsc *Reduced Surrogate Cross*

Description

Minimizes undesirable crossover results when parents have the same or many identical genes (Booker, 1987).

Usage

```
rsc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Booker L.B. (1987). Improving Search in Genetic Algorithms, in *Genetic Algorithms and Simulated Anneling*, Morgan Kaufmann Publishing.

See Also

cross, px1, kpx, sc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
rsc(parent1, parent2)
```

sax *Single Arithmetic Crossover*

Description

The Single Arithmetic Crossover (SAX) operator calculates the arithmetic mean by multiplying the parts of the Parents after a randomly determined breakpoint by a random value. Other elements remain the same.

Usage

```
sax(x1, x2, cxon, cxalfa, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxalfa	Alpha value. If no value is entered, it is randomly selected by the function in the range [0,1].
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
sax(parent1, parent2)
```

sc *Shuffle Crossover*

Description

After the SC operator determines a random cut point, it randomly shuffles both parental chromosomes and performs a single-point crossover.

Usage

```
sc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
sc(parent1, parent2)
```

selboltour *Boltzmann Tournament Selection*

Description

In the Boltzman tournament, the initial selection pressure is low. Therefore, every individual, whether low or high fitness value, has a chance to be selected. In the following generations, the selection pressure gradually increases. In other words, individuals with high fitness value are forced to be selected.

Usage

```
selboltour(fitvals, ns, selt0, selg, selgmax, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
selt0	Number, Initial temperature
selg	Current generation number
selgmax	Maximum generation number
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6, -1, 2, 4, 5)             # Fitness Values
cnames = paste0("C",1:length(fitvals))  # Chromosome Names
matpool = selboltour(fitvals, selt0=100, selg=5, selgmax=100)
cat(cnames[matpool],"\n")
matpool = selboltour(fitvals, selt0=100, selg=95, selgmax=100)
cat(cnames[matpool],"\n")
```

seldet *Deterministic Selection*

Description

Deterministic Selection is similar to Remainder Stochastic Selection. The expected value of each individual in the mating pool is calculated. Individuals are copied directly into the mating pool by the exact number of expected values. Then, sorting is done according to the fraction part of the expected values. In this case, the individuals with the highest fractions go to the top of the list to be selected. The number of individuals required to complete the mating pool to population size is selected by going from the beginning of the list to the end.

Usage

```
seldet(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6, 1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = seldet(fitvals)
cat("Selected chromosomes: ", cnames[matpool], "\n")
```

`select` *Select parents for the mating pool*

Description

The select function is a function that wraps all parent selection algorithms. It is coded for call purposes from adana main function.

Usage

```
select(selfunc, fitvals, ns, selb, selbc,
                  selc, selk, sells, selns, selps, sels, selt,
                  selt0, selw, selg, selgmax, fmin, reptype, ...)
```

Arguments

`selfunc`	Name of selection function
`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selb`	Exponent coefficent, (`selb` >= `0.0`)
`selbc`	Base of exponent
`selc`	Scaling parameter
`selk`	Power factor
`sells`	Scaling factor
`selns`	Number of Selection pressure
`selps`	Percentage of Selection, (`0.0` <= `selps` <= `1.0`)
`sels`	Selection pressure, (`1.0` <= `sels` <= `2.0`)
`selt`	Number of tournament size
`selt0`	Number, Initial temperature
`selw`	Number, Window Size
`selg`	Current generation number
`selgmax`	Maximum generation number
`fmin`	The number to subtract from all fitness values.
`reptype`	Type of Sampling, `TRUE` : `without repetitions`, `FALSE` : `with repetitions`
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Cebeci, Z. (2021). R ile Genetik Algoritmalar ve Optimizasyon Uygulamalari, 535 p. Ankara:Nobel Akademik Yayincilik.

See Also

selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, selnlrs, selers, seltrunc

Examples

```
# Create population
population = initialize(initfunc=initbin, n=10, m=8)
head(population, 5)
# Calculate fitness values
m = ncol(population)-2
fitvals = evaluate(maxone, population[,1:m])
population[,"fitval"] = fitvals
head(population, 5)
# Select parents by RWS
selidx = select(selfunc=selrws, fitvals=fitvals)
matpool = population[selidx,]
head(matpool, 5)
# Selected chromosomes
table(rownames(matpool))
```

---

selers *Exponantial Ranking Selection*

Description

The Exponantial Ranking Selection operator is a selection operator that uses probabilities obtained by exponentially weighting the ordinal numbers of individuals.

Usage

```
selers(fitvals, ns, selbc, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
selbc	Base of exponent
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, selnlrs

Examples

```
fitvals = c(6, -1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = selers(fitvals, selbc=0.1)
cat(cnames[matpool],"\n")
matpool = selers(fitvals, selbc=0.8)
cat(cnames[matpool],"\n")
```

---

selescale *Exponent Scaling*

Description

The Exponent Scaling operator is the selection operator in which the fitness values are scaled by the simulated annealing method.

Usage

```
selescale(fitvals, ns, selb, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
selb	Exponent coefficent, (selb >= 0.0)
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, seltour, seltour2

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selescale(fitvals, selb=0.1)
cat(cnames[matpool],"\n")
matpool = selescale(fitvals, selb=2)
cat(cnames[matpool],"\n")
```

---

sellrs *Linear Ranking Selection 1*

Description

The Linear Ranking Selection operator selects via probabilities obtained using ordered numbers according to their fitness values (Pohlheim, 2020).

Usage

```
sellrs(fitvals, ns, sels, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
sels	Selection pressure, (1.0 <= sels <= 2.0)
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Pohlheim, H. (2020). Tutorial for "GEATbx: Genetic and Evolutionary Algorithms Toolbox for use with MATLAB", Version 3.30, URL http://www.geatbx.com/ver_3_3/algindex-02html#P181_11564.

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)    # Fitness Values
cnames = paste0("C",1:length(fitvals))  # Chromosome Names
matpool = sellrs(fitvals)
cat(cnames[matpool],"\n")
matpool = sellrs(fitvals, sels=2)
cat(cnames[matpool],"\n")
```

---

sellrs2 *Linear Ranking Selection 2*

Description

The Linear Ranking Selection-2 operator selects via probabilities obtained using ordered numbers according to their fitness values. Selection pressure is not applied in this algorithm (Scrucca, 2013).

Usage

```
sellrs2(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Scrucca, L. (2013). GA: A package for Genetic Algorithms in R. *Journal of Statistical Software*, 53(4), 1-37.

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = sellrs2(fitvals)
cat(cnames[matpool],"\n")
```

---

sellrs3 *Linear Ranking Selection 3*

Description

The LRS operator selects through probabilities obtained using ordered numbers according to their fitness values. In this algorithm, the selection pressure can be adjusted with the s parameter.

Usage

```
sellrs3(fitvals, ns, sels, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
sels	Selection pressure, (1.0 <= sels <= 2.0)
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = sellrs3(fitvals)
cat(cnames[matpool],"\n")
matpool = sellrs3(fitvals, sels=2)
cat(cnames[matpool],"\n")
```

---

sellscale *Fitness Linear Scaling*

Description

The Fitness Linear Scaling operator scales the fitness values using a linear regression model and performs the selection (Louis, 2019).

Usage

```
sellscale(fitvals, ns, sells, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
sells	Scaling factor
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Louis, S.J. (2019). Scaling in Genetic Algorithms. URL https://www.cse.unr.edu/~sushil/class/gas/notes/scaling.pdf

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, selrscale, selrscale2, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = sellscale(fitvals)
cat(cnames[matpool],"\n")
```

`selnlrs` *Nonlinear Ranking Selection*

Description

The Nonlinear Ranking Selection is a nonlinear selection method that applies higher selection pressure than the Linear Ranking Selection (Pholheim, 1995).

Usage

```
selnlrs(fitvals, ns, selns, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selns`	Number of Selection pressure
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Pholheim, H. (1995). Ein genetischer algorithmus mit mehrfachpopulationen zur numerischen optimierung. *at-Automatisierungstechnik*, 43(3), 127-135.

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selnlrs(fitvals)
cat(cnames[matpool],"\n")
matpool = selnlrs(fitvals, selns=0.1)
cat(cnames[matpool],"\n")
```

`selpscale` *Power-law Scaling*

Description

the Power-law Scaling is a selection method in which the *k*th power of the fit values is used as the scaled fit values (Gillies, 1985).

Usage

```
selpscale(fitvals, ns, selk, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selk`	Power factor
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Gillies, A.M. (1985). Machine learning procedures for generating image domain feature detectors. PhD thesis, University of Michigan, Ann Arbor.

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selescale`, `seltour`, `seltour2`

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selpscale(fitvals, selk=1.1)
cat(cnames[matpool],"\n")
```

---

selrand *Random selection*

Description

Random selection is the process of selecting parents completely randomly from the current population, regardless of the individual's fitness values.

Usage

```
selrand(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Details

Random selection is done by simple random sampling method with replacement. Each individual has an equal chance (p = 1/n) of being selected.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, seltrunc, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, selnlrs, selers

Examples

```
fitvals = c(6, -1, 2, 4, 5)            # Fitness values
cnames = paste0("C",1:length(fitvals)) # Chromose names
matpool = selrand(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```

selrscale *Rank Scaling*

Description

The Rank Scaling is a selection method in which fitness values are scaled according to their ordinal number.

Usage

```
selrscale(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selrscale(fitvals)
cat(cnames[matpool],"\n")
```

selrscale2 *Rank Scaling 2*

Description

The Rank Scaling-2 is a selection method in which fitness values are scaled according to their ordinal number. Selection pressure can be adjusted by the user.

Usage

```
selrscale2(fitvals, ns, sels, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
sels	Scaling factor, (1.0 <= sels <= 2.0)
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selrscale2(fitvals)
cat(cnames[matpool],"\n")
matpool = selrscale2(fitvals, sels=2)
cat(cnames[matpool],"\n")
```

`selrss` *Remainder Stochastic Selection*

Description

The fitness probability of individuals is multiplied by the population size to calculate the number of times the individual will reproduce in the mating pool, ie the expected number of copies. The expected number of copies is a fractional number. An exact fraction of the expected number of copies of the individual is sent to the mating pool. It is also determined whether it can go back to the mating pool for the fraction part (Brindle, 1981).

Usage

```
selrss(fitvals, ns, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Brindle, A. (1981). Genetic algorithms for function optimization. PhD thesis, University of Alberta.

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2` , `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`

Examples

```
fitvals = c(6, 1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = selrss(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```

selrswrp *Random selection with replacement and proportion*

Description

Random selection is made by simple random sampling method with replacements, based on the fitness values of individuals. Each individual has the chance to be selected proportionally to their fitness value.

Usage

```
selrswrp(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, seltrunc, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, selnlrs, selers

Examples

```
fitvals = c(6, 1, 2, 4, 5)              # Fitness Values
cnames = paste0("C",1:length(fitvals))  # Chromosome names
matpool = selrswrp(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```

selrws *Roulette wheel selection 1*

Description

This function provides the opportunity to take more than one place in the mating pool in proportion to the fitness value of each individual.

Usage

```
selrws(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, seltrunc, selrws2, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, selnlrs, selers

Examples

```
fitvals = c(6, 1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = selrws(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```

`selrws2` *Roulette wheel selection 2*

Description

This function provides the opportunity to take more than one place in the mating pool in proportion to the fitness value of each individual.

Usage

```
selrws2(fitvals, ns, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `seltrunc`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`, `selnlrs`, `selers`

Examples

```
fitvals = c(6, 1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = selrws(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```

selsscale *Sigma Scaling*

Description

Sigma Scaling is based on the mean rather than the worst fitness value as in Window Scaling. In Sigma Scaling, an individual's fitness is a function of the population mean and population standard deviation (Forrest, 1985; Goldberg, 1989).

Usage

```
selsscale(fitvals, ns, selc, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
selc	Scaling parameter
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Forrest, S. (1985). Documentation of prisoner's dilemma and norms programs that use the genetic algorithm. Technical report, University of Michigan.

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, selsus, seldet, selwscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selsscale(fitvals, selc=2)
cat(cnames[matpool],"\n")
```

`selsscale2` *Sigma scaling 2*

Description

Sigma Scaling is based on the mean rather than the worst fitness value as in Window Scaling. In Sigma Scaling, an individual's fitness is a function of the population mean and population standard deviation. In this approach, if the scaled value is less than zero, it is set to zero.

Usage

```
selsscale2(fitvals, ns, selc, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selc`	Scaling parameter
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(fitvals))
matpool = selsscale2(fitvals)
cat(cnames[matpool],"\n")
```

selsus *Stochastic Universal Selection*

Description

The Stochastic Universal Selection is the Roulette Wheel Selection method with multiple winning points.

Usage

```
selsus(fitvals, ns, ...)
```

Arguments

fitvals	Vector of fitness values belonging to individuals
ns	Number of individuals to be selected
...	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

select, selrand, selrswrp, selrws, selrws2, selrss, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2

Examples

```
fitvals = c(6, 1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = selsus(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```

`seltour` *Tournament Selection*

Description

The best one is selected in the group consisting of t individuals selected by random sampling with or without replacement from the current population (Smith et.al, 1991).

Usage

```
seltour(fitvals, ns, selt, reptype, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selt`	Number of tournament size
`reptype`	Type of Sampling, `TRUE` : `without repetitions`, `FALSE` : `with repetitions`
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Smith, R.E., Goldberg, D.E. and Earickson, J.A. (1991). SGA-C: A C-language implementation of a simple gewnetic algorithm. Technical report 91002, Illinois Genetic Algorithms Laboratory, Urbana, IL, USA.

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour2`

Examples

```
selt = 2                              # Size of tournament
fitvals = c(6, -1, 2, 4, 5)           # Fitness values
cnames = paste0("C",1:length(fitvals)) # Chromosome names
matpool = seltour(fitvals, selt=selt)
cat(cnames[matpool],"\n")
```

`seltour2` *Tournament Selection 2*

Description

Each individual is given a chance to participate in the tournament at least once in selection by tournament in this function. For this reason, individuals participating in the tournament cannot participate in another tournament, but after the tournament of all individuals is completed, they can get a chance to participate in another tournament (Nicolau, 2009).

Usage

```
seltour2(fitvals, ns, selt, reptype, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selt`	Number of tournament size
`reptype`	Type of Sampling, `TRUE : without repetitions`, `FALSE : with repetitions`
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Nicolau, M. (2009). Application of a simple binary genetic algorithm to a noiseless testbed benchmark. In *Proc. genetic and Evolutionary Computation Conf. (GECCO)*, Montreal, Canada.

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`, `selnlrs`, `selers`, `seltrunc`, `select`

Examples

```
selt = 2                    # Size of tournament
fitvals = c(6, -1, 2, 4, 5) # Fitness values
cnames = paste0("C",1:length(fitvals)) # Chromosome names
matpool = seltour2(fitvals, selt=selt)
cat(cnames[matpool],"\n")
```

`seltrunc` *Truncation Selection*

Description

Individuals in the population are ranked according to their fitness value and individuals with higher fitness value than a determined threshold value are included in the mating pool.

Usage

```
seltrunc(fitvals, ns, selps, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`selps`	Percentage of Selection, (`0.0` <= `selps` <= `1.0`)
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`, `selnlrs`, `selers`

Examples

```
fitvals = c(6, -1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = seltrunc(fitvals, selps=0.60)
cat(cnames[matpool],"\n")
```

`selwscale` *Window Scaling*

Description

Window Scaling is a method based on subtracting the worst fitness value from the other fitness values. In this case, since the scaled values of the worst fit individuals will be 0, these individuals will not be given a chance to be selected.

Usage

```
selwscale(fitvals, ns, fmin, ...)
```

Arguments

`fitvals`	Vector of fitness values belonging to individuals
`ns`	Number of individuals to be selected
`fmin`	The number to subtract from all fitness values.
`...`	Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

`select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selsus`, `seldet`, `selrss`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`

Examples

```
fitvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
fmin = min(fitvals)
cnames = paste0("C",1:length(fitvals))
matpool = selwscale(fitvals, fmin=fmin)
cat(cnames[matpool],"\n")
fitvals = fitvals[matpool]
fitvals
matpool = selwscale(fitvals, fmin=fmin)
cat(cnames[matpool],"\n")
fitvals = fitvals[matpool]
fitvals
fmin = min(fitvals)
matpool = selwscale(fitvals, fmin=fmin)
cat(cnames[matpool],"\n")
```

show *Function to visualize iteration results*

Description

The show function provides access to user-defined visualization functions.

Usage

```
show(monitorfunc, g, genfits, objective, x, ...)
```

Arguments

monitorfunc	Monitoring function
g	Generation number
genfits	A matrix for fitness values
objective	Type of optimization. "min" or "max"
x	...
...	Further arguments passed to or from other methods.

Value

NA

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

monprogress

Examples

```
n = 100
genfits = matrix(NA, nrow=n, ncol=5)
genfits[1,3] = 50
objective = "max"
monitorfunc = monprogress
for(i in 1:(n-1)){
  g=i
  show(monitorfunc,  g=g, genfits=genfits,
    objective=objective, x=NULL)
  genfits[g+1, 3] = genfits[g, 3] + runif(1, -2, 5)
}
```

shufmut *Shuffle Mutation*

Description

Shuffle Mutation works by randomly shuffling the values in a randomly selected subset of the chromosome (Syswerda, 1991).

This operator is used in problems with permutation encoding.

Usage

```
shufmut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutrange	A vector. The numbers of begining and ending of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Syswerda, G. (1991). Schedule optimization using genetic algorithms. *Handbook of Genetic Algorithms*.

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
shufmut(offspring)
```

smc *Sinusoidal Motion Crossover (SMC)*

Description

The proposed algorithm with the name of SMC is a simple algorithm that works deterministic and alternatively (Kumar & Panneerselvam, 2017).

Usage

```
smc(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Varun Kumar, S.G. and Panneerselvam R. (2017). A Study of Crossover Operators for Genetic Algorithms to Solve VRP and its Variants and New Sinusoidal Motion Crossover Operator. *International Journal of Computational Intelligence Research*. Vol 13, Number 7, pp. 1717-1733

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx

Examples

```
parent1 = c(1, 2, 3, 4, 5, 6, 7, 8)
parent2 = c(4, 6, 7, 3, 2, 1, 8, 6)
smc(parent1, parent2)
```

spherex *Sphere Crossover*

Description

Sphere Crossover is an operator performed by applying Sphere equality to parent chromosomes. It generates one offspring per each cross.

Usage

```
spherex(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
spherex(parent1, parent2)
```

ssrfamtour *Replacement function via family tournament*

Description

The two most compatible between the two parents and their offspring are added to the new generation population, while those with low fitness are discarded (Sivanandam et.al., 2007).

Usage

```
ssrfamtour(parpop, offpop, reppars, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
reppars	A vector. Indices of the parents
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Sivanandam et.al., (2007).

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrgenitor, ssrx

`ssrgenitor` *Genitor replacement function*

Description

The offspring obtained by mating two randomly selected parents from the mating pool is placed in the place of the worst individual in the current population (Whitley, 1988).

Usage

```
ssrgenitor(parpop, offpop, ...)
```

Arguments

`parpop`	A matrix. Parent population
`offpop`	A matrix. Offspring population
`...`	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Whitley, L.D. (1988). GENITOR: A different genetic algorithm. In *Proc. of the Rocky Mountain conf. on artificial intellegence*, pp. 118-130.

See Also

`grdelall`, `elitism`, `grmuplambda`, `grmuplambda2`, `grmuplambda3`, `grmuplambda4`, `grmuvlambda`, `grrobin`, `ssrmup1`, `ssrfamtour`, `ssrx`

ssrmup1 *Mu+1 replacement function*

Description

Two randomly selected parents from the mating pool are mated to produce one or more offspring. The fit value of an individual randomly selected from the population is compared with the offspring with the highest fitness value. If the fitness value of the offspring is higher, the offspring is replaced with the individual.

Usage

```
ssrmup1(parpop, offpop, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrgenitor, ssrfamtour, ssrx

ssrx *Mixed replacement function*

Description

The offspring with the best fitness value takes the place of an individual randomly selected from among the individuals excluding their parents and the individual with the worst fitness value in the population.

Usage

```
ssrx(parpop, offpop, reppars, ...)
```

Arguments

parpop	A matrix. Parent population
offpop	A matrix. Offspring population
reppars	A vector. Indices of the parents
...	Further arguments passed to or from other methods.

Value

A matrix. Population of the new generation.

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvlambda, grrobin, ssrmup1, ssrgenitor, ssrfamtour

---

swapmut *Swap Mutation*

Description

SM is the reciprocal exchange of the values of two randomly selected genes on the chromosome (Banzhaf, 1990).

This operator is used in problems with permutation or binary encoding.

Usage

```
swapmut(y, ...)
```

Arguments

y	A vector. Chromosome of the offspring
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	A vector. The numbers of the mutated genes.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Banzhaf, W. (1990). The "molecular" traveling salesman. *Biological Cybernetics*, 64(1), 7-14.

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, unimut, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
swapmut(offspring)
```

---

terminate *Termination Control Function*

Description

The function of terminating the genetic algorithm

Usage

```
terminate(tercrit, maxiter, objective, t, genfits,
                    fitvals, objval, optdif, rmcnt, rmdif, abdif, mincv,
                    sddif, rangedif, simlev, phidif, meandif, bestdif,
                    stime, maxtime)
```

Arguments

tercrit	A vector. Indications of termination criteria.
maxiter	Maximum iteration
objective	?????
t	Generation number
genfits	A matrix. Best fitness of each generation
fitvals	Fitness values of current generation
objval	Global optimum value
optdif	Difference from global optimum value
rmcnt	k value for minimum difference of the mean of the last k best fitness values.
rmdif	The minimum difference between the mean of the last k best fitness values and the best fitness value in the current generation.
abdif	Minimum difference between best fitness value and mean of fitness values
mincv	Minimum coefficient of variance
sddif	The minimum difference between the last two standard deviations.

`rangedif`	Minimum and maximum difference (range of change)
`simlev`	Similarity percentage of fitness values
`phidif`	Phi convergence
`meandif`	The minimum difference between the last two fitness values
`bestdif`	Percentage of difference between the last two best fitness values
`stime`	System time saved before starting GA
`maxtime`	Maximum running time

Value

Termination criterion

0 : No termination

1 : Maximum iteration

2 : Reaching the global optimum value

3 : Converging the global optimum

4 : The minimum difference between the last two fitness values

5 : Percentage of difference between the last two best fitness values

6 : Minimum difference of the mean of the last k best fitness values

7 : Minimum difference between best fitness value and mean of fitness values

8 : The minimum difference between the last two standard deviations.

9 : Minimum and maximum difference (range of change)

10: Minimum coefficient of variance

11: Phi convergence

12: Similarity percentage of fitness values

13: Maximum running time

Author(s)

Zeynel Cebeci & Erkut Tekeli

`unimut` *Uniform Mutation*

Description

The Random Resetting Mutation operator replaces the value of a randomly selected gene with a randomly selected value between the allowed limits for that gene (Michalewicz, 1994).

This operator is used for value encoded (integer or real number) chromosomes.

Usage

```
unimut(y, lb, ub, ...)
```

Arguments

y	A vector. Chromosome of the offspring
lb	A vector. Lower bounds of genes
ub	A vector. Upper bounds of genes
...	Further arguments passed to or from other methods.

Value

mutant	A vector. Chromosome of the offspring
mutgen	The number of the mutated gene.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Michalewicz, . (1994).

See Also

mutate, bitmut, randmut, randmut2, randmut3, randmut4, boundmut, nunimut, nunimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut

Examples

```
lb = c(2, 1, 3, 1, 0, 4)
ub = c(10, 15, 8, 5, 6, 9)
offspring = c(8, 6, 4, 1, 3, 7)
unimut(offspring, lb, ub)
```

---

upmx *Uniform Partial Mapped Crossover*

Description

Uniform Partial Mapped Crossover (UPMX) is a crossover operator for permutation encoded chromosomes. Parent1 is cloned into Offspring1. A random point v1 is chosen. The gene at point v1 in Parent2 is determined. The v2 point carrying this gene is determined in Offspring1. The genes at v1 and v2 are swapped. These processes are repeated k times (Migkikh et.al., 1996).

Usage

```
upmx(x1, x2, cxon, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Migkikh, V.V., Topchy, A.A., Kureichick, V.M. and Tetelbaum, A.Y. (1996). Combined Genetic and local search algorithm for the quadratic assignment problem. In *Proc. of IC on Evolutionary Computation and Its Applications, EvCA*, 96, 335-341.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(0, 8, 4, 5, 6, 7, 1, 2, 3, 9)
parent2 = c(6, 7, 1, 2, 4, 8, 3, 5, 9, 0)
upmx(parent1, parent2)
```

---

ux *Uniform crossover 1*

Description

In a uniform crossover, the number of crossover points is not fixed and evaluates each gene independently (De Jong & Spears, 1991). In other words, it generalizes multi-point crossover as each gene locus is viewed as a potential crossover point.

Usage

```
ux(x1, x2, cxon, cxps, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxps	It determines the rate of gene exchange between the chromosomes of the parents.
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

De Jong, K.A. and Spears, W. (1991). On the virtues of parameterized uniform crossover. In *Proc. of the 4th Int. Conf. on Genetic Algorithms*. Morgan Kaufman Publishers.

See Also

cross, px1, kpx, sc, rsc, hux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
ux(parent1, parent2, cxon=3)
```

---

ux2 *Uniform Crossover 2*

Description

In a uniform crossover, the number of crossover points is not fixed and evaluates each gene independently (De Jong & Spears, 1991). In other words, it generalizes multi-point crossover as each gene locus is viewed as a potential crossover point.

Usage

```
ux2(x1, x2, cxon, cxps, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxps	It determines the rate of gene exchange between the chromosomes of the parents.
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

De Jong, K.A. and Spears, W. (1991). On the virtues of parameterized uniform crossover. In *Proc. of the 4th Int. Conf. on Genetic Algorithms*. Morgan Kaufman Publishers.

See Also

cross, px1, kpx, sc, rsc, hux, ux, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
ux2(parent1, parent2, cxon=2)
```

---

wax *Whole Arithmetic Crossover*

Description

New offspring are produced by applying an arithmetic mean to all of the parents' chromosomes. (Davis, 1985; Back et.al, 1991; Michalewicz & Janikov, 1991; Michalewicz, 1992; Michalewicz, 1995).

Usage

```
wax(x1, x2, cxon, cxalfa, ...)
```

Arguments

x1	A vector. It contains the chromosomal information of parent-1.
x2	A vector. It contains the chromosomal information of parent-2.
cxon	Number of offspring to be generated as a result of crossover
cxalfa	Alpha value. If no value is entered, it is randomly selected by the function in the range [0,1].
...	Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

Davis, L. (1985). Aplaying adaptive algorithms to epistatics domains. In *Proc. of the Int. Joint Conf. on Artificial Intellengence*, Vol. 85, pp. 162-164.

Back, T., Hoffmeister, F. and Schwefel, H.P. (1991). A survey of evolution strategies. In *Proc. of the 4th Int. Conf. on Genetic Algorithms*, pp. 2-9. Morgan Kaufmann.

Michalewicz, Z. and Janikov, S.J. (1991). Genetic algorithms for numerical optimization. *Statistics and Computing*, 1(2), 75-91.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. Berlin-Heidelberg: Springer Verlag.

Michalewicz, Z. (1995). Genetic algorithms, numerical optimization and constraints. In *Proc. of the 4th Int. Conf. on Genetic Algorithms*. pp. 151-158. Morgan Kaufmann.

See Also

cross, px1, kpx, sc, rsc, hux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```
parent1 = c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4)
wax(parent1, parent2)
```

Index