Package ‘adana’

February 23, 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 ................................................................. 5
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>adana</td>
<td>5</td>
</tr>
<tr>
<td>adana1</td>
<td>12</td>
</tr>
<tr>
<td>adana2</td>
<td>13</td>
</tr>
<tr>
<td>adana3</td>
<td>14</td>
</tr>
<tr>
<td>atc</td>
<td>15</td>
</tr>
<tr>
<td>ax</td>
<td>16</td>
</tr>
<tr>
<td>bestsol</td>
<td>17</td>
</tr>
<tr>
<td>bin2gray</td>
<td>17</td>
</tr>
<tr>
<td>bin2int</td>
<td>18</td>
</tr>
<tr>
<td>bitmut</td>
<td>19</td>
</tr>
<tr>
<td>blxa</td>
<td>20</td>
</tr>
<tr>
<td>blxab</td>
<td>21</td>
</tr>
<tr>
<td>boundmut</td>
<td>22</td>
</tr>
<tr>
<td>bsearchmut1</td>
<td>23</td>
</tr>
<tr>
<td>bsearchmut2</td>
<td>24</td>
</tr>
<tr>
<td>bx</td>
<td>25</td>
</tr>
<tr>
<td>calcM</td>
<td>26</td>
</tr>
<tr>
<td>cpc</td>
<td>27</td>
</tr>
<tr>
<td>cross</td>
<td>28</td>
</tr>
<tr>
<td>cx</td>
<td>29</td>
</tr>
<tr>
<td>dc</td>
<td>30</td>
</tr>
<tr>
<td>decode</td>
<td>31</td>
</tr>
<tr>
<td>decode4int</td>
<td>32</td>
</tr>
<tr>
<td>decodepop</td>
<td>33</td>
</tr>
<tr>
<td>disc</td>
<td>34</td>
</tr>
<tr>
<td>dismut</td>
<td>35</td>
</tr>
<tr>
<td>ebx</td>
<td>36</td>
</tr>
<tr>
<td>eclc</td>
<td>37</td>
</tr>
<tr>
<td>elitism</td>
<td>38</td>
</tr>
<tr>
<td>elx</td>
<td>39</td>
</tr>
<tr>
<td>encode</td>
<td>40</td>
</tr>
<tr>
<td>encode4int</td>
<td>41</td>
</tr>
<tr>
<td>encodepop</td>
<td>42</td>
</tr>
<tr>
<td>erx</td>
<td>43</td>
</tr>
<tr>
<td>evaluate</td>
<td>44</td>
</tr>
<tr>
<td>findoptima</td>
<td>45</td>
</tr>
<tr>
<td>fixpcmut</td>
<td>46</td>
</tr>
<tr>
<td>gaussmut</td>
<td>47</td>
</tr>
<tr>
<td>gaussmut2</td>
<td>48</td>
</tr>
<tr>
<td>gaussmut3</td>
<td>49</td>
</tr>
<tr>
<td>geomx</td>
<td>50</td>
</tr>
<tr>
<td>gray2bin</td>
<td>51</td>
</tr>
<tr>
<td>gray2bin2</td>
<td>52</td>
</tr>
<tr>
<td>grdelall</td>
<td>53</td>
</tr>
<tr>
<td>grmuplambda</td>
<td>53</td>
</tr>
<tr>
<td>grmuplambda2</td>
<td>54</td>
</tr>
<tr>
<td>grmuplambda3</td>
<td>55</td>
</tr>
<tr>
<td>grmuplambda4</td>
<td>56</td>
</tr>
</tbody>
</table>
R topics documented:

- grmu(lambda) .................................................. 56
- grrobin .......................................................... 57
- hcn ................................................................. 58
- hgaoptim ......................................................... 59
- hgaoptimx ....................................................... 60
- hgaroi ............................................................ 61
- hx ................................................................. 62
- icx ................................................................. 63
- ilmdhc ............................................................ 64
- initbin ............................................................ 65
- initialize .......................................................... 66
- initnorm ............................................................ 67
- initperm ............................................................. 68
- initval ............................................................... 69
- insmut .............................................................. 70
- insswapmut ......................................................... 71
- int2bin .............................................................. 72
- invdismut .......................................................... 73
- invmut .............................................................. 74
- invswapmut .......................................................... 75
- kpx ................................................................. 76
- lapx ................................................................. 77
- lax ................................................................. 78
- leitingzhi ........................................................... 79
- maxone ............................................................. 80
- maxone1 ............................................................ 80
- maxone2 ............................................................ 81
- minone .............................................................. 82
- monprogress ........................................................ 83
- mpmx ............................................................... 84
- mpx ................................................................. 85
- mutate ............................................................... 86
- mx ................................................................. 87
- nunimut ............................................................. 88
- nunimut2 ............................................................ 89
- ox ................................................................. 90
- ox2 ................................................................. 91
- pbx ................................................................. 92
- pbx2 ............................................................... 93
- plotfitness ......................................................... 94
- pmx ................................................................. 95
- powmut ............................................................. 96
- powmut2 ............................................................ 97
- px1 ................................................................. 98
- randmut ............................................................ 99
- randmut2 ........................................................... 100
- randmut3 ........................................................... 101
- randmut4 ........................................................... 102
R topics documented:

raoc ................................................................. 103
rrc ................................................................. 104
rsc ................................................................. 105
sax ................................................................. 106
sc ................................................................. 107
selboltour ...................................................... 108
seldet ............................................................. 109
select ............................................................. 110
selers ............................................................. 111
selescale .......................................................... 112
sellsrs ............................................................ 113
sellsrs2 ........................................................... 114
sellsrs3 ........................................................... 115
sellscale .......................................................... 116
selnlrs ............................................................ 117
selpscale .......................................................... 118
selrand ............................................................ 119
selrscale .......................................................... 120
selrscale2 ......................................................... 121
selrsrs ............................................................. 122
selrswrp ........................................................... 123
selrws ............................................................. 124
selrws2 ............................................................ 125
sellscale .......................................................... 126
selsscale2 .......................................................... 127
selssus ............................................................ 128
seltour ............................................................ 129
seltour2 ........................................................... 130
seltrunc ........................................................... 131
selwscale .......................................................... 132
show ............................................................... 133
shufmut ............................................................ 134
smc ................................................................. 135
spherex ............................................................ 136
ssrfamtour ......................................................... 137
ssrgenitor .......................................................... 138
ssrmup1 ............................................................ 139
ssrx ................................................................. 139
swapmut ........................................................... 140
terminate .......................................................... 141
unimut ............................................................. 142
upmx ............................................................... 143
ux ................................................................. 144
ux2 ................................................................. 145
wax ................................................................. 146

Index 148
**Description**

The package **adana** provides the functions related to the hybrid 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 efficiencies 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**


**See Also**

adana
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, seips = 0.5, selw = 2, retype = FALSE, cxpc = 0.9, cxpc2 = 0.8, cxxn = 2, cxxk = 2, cxps = 0.5, cxxa = 0, cxxb = 0.15, cxalpha = 1, cxalpha = 0.5, mutpm = 0.05, mutpm2 = 0.2, mutb = 2, mutpaw = 2, mutq = 0.5, mutmy = c(), mutedy = c(), adap = 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, rmidf = 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"
adaptfunc  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.
seins     Polynomial 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 selrs, selrs3 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.

**cxf**

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

**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.

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

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

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".

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The 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 `hgfunc` 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**


**See Also**

- GA Operators: `initialize`, `evaluate`, `terminate`, `select`, `cross`, `mutate`
- Initialize Functions: `initbin`, `initval`, `initperm`, `initnorm`
- Selection Functions: `selrand`, `selrswrp`, `selrws`, `selrws2`, `selsus`, `seldet`, `selwscale`, `selsscale`, `seลsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltscale`, `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`, `blxb`, `lapx`, `elx`, `geon`, `sphere`, `pmx`, `mpmx`, `upmx`, `ox`, `ox2`, `mpx`, `erx`, `px`, `px2`, `cx`, `icx`, `smc`
- Mutation Functions: `bitmut`, `randmut`, `randmut2`, `randmut3`, `randmut4`, `unimut`, `boundmut`, `unimut`, `unimut2`, `pomut`, `pomut2`, `gaussmut`, `gaussmut2`, `gaussmut3`, `bsearchmut`, `bsearchmut2`, `swapmut`, `invmut`, `shufmut`, `insmut`, `dismut`, `invswapmut`, `insswapmut`, `invdismut`
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

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


See Also

fixpcmut, ilmdhc, adana2, leitingzhi, adana3

Examples

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

Description

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

Usage

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

Arguments

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

<table>
<thead>
<tr>
<th>pc</th>
<th>Crossover rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm</td>
<td>Mutation rate</td>
</tr>
</tbody>
</table>

Author(s)

Zeynel Cebeci & Erkut Tekeli

References


See Also

fixpcmut, ilmdhc, adana1, leitingzhi, adana3

Examples

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

---

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

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

Arguments

<table>
<thead>
<tr>
<th>fitvals</th>
<th>A vector. Fitness values of current generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Current generation</td>
</tr>
<tr>
<td>gmax</td>
<td>Maximum generation</td>
</tr>
<tr>
<td>cxpc</td>
<td>Crossover rate. 0 &lt;= cxpc &lt;= 1</td>
</tr>
<tr>
<td>mutpm</td>
<td>Mutation rate. 0 &lt;= mutpm &lt;= 1</td>
</tr>
<tr>
<td>adapc</td>
<td>Adaptation threshold for crossover rate. 0 &lt;= adapc &lt;= 1. default is 0.05</td>
</tr>
<tr>
<td>adapd</td>
<td>Adaptation threshold for mutation rate. 0 &lt;= adapb &lt;= 1. default is 0.05</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments passed to or from other methods.</td>
</tr>
</tbody>
</table>
atc

Value

<table>
<thead>
<tr>
<th>pc</th>
<th>Crossover rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm</td>
<td>Mutation rate</td>
</tr>
</tbody>
</table>

Author(s)

Zeynel Cebeci & Erkut Tekeli

References


See Also

fixpcmut, ilmdhc, adana1, adana2, leitingzhi

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


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, ppmx, upmx, ox, ox2, mp, erx, pbx, pbx2, cx, icx, smc

Examples

```r
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
ax(parent1, parent2)
```

---

**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**

```r
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**

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, pnx, 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**

```r
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.
**bitmut**

**Author(s)**
Zeynel Cebeci & Erkut Tekeli

**See Also**

*int2bin*

**Examples**

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

---

## 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**

```r
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*, *invsdismut*

**Examples**

```r
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

```r
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


### 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

```r
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)
```
Description

Blended crossover-αβ 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, pnx, 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)
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

```r
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**

```r
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**


**See Also**

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

```r
dooffspring = 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**

```r
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**


**See Also**

- `mutate`, `bitmut`, `randmut`, `randmut2`, `randmut3`, `randmut4`, `unimut`, `boundmut`, `numimut`, `numimut2`, `powmut`, `powmut2`, `gaussmut`, `gaussmut2`, `gaussmut3`, `bsearchmut1`, `swapmut`, `invmut`, `shufmut`, `insmut`, `dismut`, `invswapmut`, `insswapmut`, `invdismut`
bx

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


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)
```
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


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, mpox, 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  

**Description**

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

**Usage**

```r
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**


**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, px, px2, cx, icx, smc
Examples

```r
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

```r
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

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, spherox, 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, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherox, 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)

Example

x = c(0,1,0,0,0,1,1)
dedecode(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 an integer number.

**Usage**

```r
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**

```r
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

```r
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
enccpop = encodepop(pop, lb=lb, ub=ub, eps=eps)
pop = enccpop$binmat
m = enccpop$m
```
disctype = 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

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, 0, 1, 0, 0, 1, 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**

\[ \text{dismut}(y, \ldots) \]

**Arguments**

- \( y \): A vector. Chromosome of the offspring
- \( \ldots \): Further arguments passed to or from other methods.

**Value**

- \( \text{mutant} \): A vector. Chromosome of the offspring
- \( \text{mutrange} \): A vector. The numbers of begining and ending of the mutated genes.
- \( r \): The number of insertion location.

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**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**

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

\[
\text{ebx}(x1, x2, lb, ub, \text{cxon}, \text{cxalfa}, \ldots)
\]

**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 \([0,1]\).
- **\ldots**: Further arguments passed to or from other methods.

**Value**

A matrix containing the generated offsprings.

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**See Also**

cross, px1, kp1, 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, mpvx, erx, pbx, pbx2, cx, icx, smc
Examples

\[
\begin{align*}
    \text{lb} &= c(0, 0, 0, 0, 0, 0) \\
    \text{ub} &= c(2, 3, 1, 2, 4, 3) \\
    \text{parent1} &= c(1.1, 1.6, 0.0, 1.1, 1.4, 1.2) \\
    \text{parent2} &= c(1.2, 0.0, 0.0, 1.5, 1.2, 1.4) \\
    \text{ebx(parent1, parent2, lb, ub)}
\end{align*}
\]

Description

Exchange/Linkage Crossover (EC, 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

\[
\text{eclc}(x1, x2, \text{cxon}, \ldots)
\]

Arguments

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

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References


See Also

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

```r
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**

```r
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**


**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, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, geomx, spere, pxm, mpmx, umpmx, oxa, oxa2, mpnx, erx, pbx, pbx2, cx, icx, smc

Examples

```r
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**

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

```r
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**

```r
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])
```
erx

pop = cbind(x, y, w)
pop
cencpop = 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**


**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, sphex, pmx, mpmx, upmx, ox, ox2, mp, 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 individual 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 an 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


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**

```r
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
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


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

```r
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)
```

---

**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**

```r
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`, `insswapmut`, `insswapmut`, `invdismut`

**Examples**

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

```r
gemx(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


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, mp, erx, pbx, pbx2, cx, icx, smc

Examples

```r
parent1 = c(1.1, 1.6, 0.8, 1.1, 1.4, 1.2)
parent2 = c(1.2, 0.8, 0.8, 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**

```r
g = c(1,1,1,0)
g21 = gray2bin(g) # returns 1011

g = c(1,1,1,1)
g21 = gray2bin(g) # 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**

```r
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**


**See Also**

`bin2gray`, `gray2bin`

**Examples**

```r
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parpop</td>
<td>A matrix. Parent population</td>
</tr>
<tr>
<td>offpop</td>
<td>A matrix. Offspring population</td>
</tr>
</tbody>
</table>

Value


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+Lambda 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parpop</td>
<td>A matrix. Parent population</td>
</tr>
<tr>
<td>offpop</td>
<td>A matrix. Offspring population</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments passed to or from other methods.</td>
</tr>
</tbody>
</table>
grmuplambda2

Value

Author(s)
Zeynel Cebeci & Erkut Tekeli

References

See Also
grdelall, elitism, grmuplambda2, grmuplambda3, grmuplambda4, grmuvtlambda, grrobin, ssrmup1, ssrmup1, ssrgenitor, ssrfamtour, ssrx

grmuplambda2  Mu+Lambda replacement function 2 (delete the worst \( \lambda \))

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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parpop</td>
<td>A matrix. Parent population</td>
</tr>
<tr>
<td>offpop</td>
<td>A matrix. Offspring population</td>
</tr>
<tr>
<td>lambda</td>
<td>Number of individuals renewed in the population</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments passed to or from other methods.</td>
</tr>
</tbody>
</table>

Value

Author(s)
Zeynel Cebeci & Erkut Tekeli


grmuplambda3

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 $\lambda$ best fit offspring are replaced by $\lambda$ parents randomly selected from the current parent population.

**Usage**

```r
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**


**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**See Also**

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

In the current population, randomly selected $\lambda$ parents are replaced by randomly selected $\lambda$ 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


Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuvlamba, grrobin, srmup1, ssrgenitor, ssrfamtour, ssrx

---

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**


**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**See Also**

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

---

**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**


**Author(s)**

Zeynel Cebeci & Erkut Tekeli
**See Also**

grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvtake, ssrmup1, ssrgenitor, ssrfamto, 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**

\[
hc(x1, x2, \text{fitfunc}, \text{cxon}, \ldots)
\]

**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
- **\ldots**: Further arguments passed to or from other methods.

**Value**

A matrix containing the generated offsprings.

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**See Also**

cross, px1, kpx, sc, rsc, hux, 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, mp, erx, pbx, pbx2, cx, icx, smc
**Examples**

```r
fittfunc = 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, fittfunc)
```

---

**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**

```r
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**

See Also

\texttt{hgaoptimx}, \texttt{hgaroi}

Examples

```r
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
```

Description

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

Usage

\texttt{hgaoptimx(genpop, fitfunc, hgaparams,}
\texttt{ 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.
  - \texttt{w}: individuals with the worst fitness value
  - \texttt{b}: individuals with the best fitness value
  - \texttt{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.
hgaroi

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**See Also**

hgaoptim, hgaroi

**Examples**

```r
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",
                 popim=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


See Also

hgaoptim, hgaoptimx

Examples

```r
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
```
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


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, eix, geomx, spherex, pmx, mpmx, upmx, ox, ox2, mp, 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)
Improved Cycle Crossover (ICX)

Description

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

Usage

\[ \text{icx}(x_1, x_2, \text{cxon}, \ldots) \]

Arguments

- \( x_1 \): A vector. It contains the chromosomal information of parent-1.
- \( x_2 \): A vector. It contains the chromosomal information of parent-2.
- \( \text{cxon} \): Number of offspring to be generated as a result of crossover
- \( \ldots \): Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References


See Also

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

Examples

```r
cparent1 = c(3, 4, 8, 2, 7, 1, 6, 5)
cparent2 = c(4, 2, 5, 1, 6, 8, 3, 7)
icx(pparent1, pparent2)
```
**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**


**See Also**

fixpcmut, adana1, adana2, leitingzhi, adana3

**Examples**

```r
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
```
\[ \text{nm} = \text{round}(\text{pm} \times N) \]
\[ \text{nm} = \text{ifelse (!nm, 1, nm)} \]

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

\begin{verbatim}
 n = 20  #Population size (number of chromosemes)
m = 5   #Number of gene (chromosome length)
population = initbin(n, m)
head(population, 4)
tail(population, 4)
\end{verbatim}

**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

\begin{verbatim}
initpop = initialize(initfunc=initbin, n=6, m=4)
initpop
\end{verbatim}
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**

```r
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**

```r
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
**insmut**

**See Also**

initbin, initperm, initnorm, initialize

**Examples**

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

**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**

```r
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**

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

It is a mutation operator that combines insertion and inversion mutation.
This operator is used in problems with permutation encoding.

Usage

\texttt{insswapmut(y, ...)}

Arguments

\texttt{y} \hspace{1cm} \text{A vector. Chromosome of the offspring}
\texttt{...} \hspace{1cm} \text{Further arguments passed to or from other methods.}

Value

\texttt{mutant} \hspace{1cm} \text{A vector. Chromosome of the offspring}
\texttt{mutgen} \hspace{1cm} \text{A vector. The numbers of beginning and ending of the mutated genes.}

Author(s)

Zeynel Cebeci & Erkut Tekeli

See Also

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

Examples

\texttt{offspring = c(1, 2, 3, 4, 5, 6, 7, 8, 9)}
\texttt{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 beginning and ending of the mutated genes.
r The number of insertion 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 beginning and ending of the mutated genes.
invswapmut

Author(s)
Zeynel Cebeci & Erkut Tekeli

References

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 beginning 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)
invsawmpmut(offspring)

---

des

**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, mp, erx, pbx, pbx2, cx, icx, smc

Examples

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

\[ \text{lapx}(x_1, x_2, c_{xon}, c_{xa}, c_{xb}, \ldots) \]

Arguments

- \( x_1 \): A vector. It contains the chromosomal information of parent-1.
- \( x_2 \): A vector. It contains the chromosomal information of parent-2.
- \( c_{xon} \): Number of offspring to be generated as a result of crossover.
- \( c_{xa} \): Location Parameter
- \( c_{xb} \): Scale Parameter. \((c_{xb} > 0)\)
- \( \ldots \): Further arguments passed to or from other methods.

Value
A matrix containing the generated offsprings.

Author(s)
Zeynel Cebeci & Erkut Tekeli

References

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

Examples

\[ \text{parent1} = \mathbf{c}(1.1, 1.6, 0.0, 1.1, 1.4, 1.2) \]
\[ \text{parent2} = \mathbf{c}(1.2, 0.0, 0.0, 1.5, 1.2, 1.4) \]
\[ \text{lapx(parent1, parent2, cxon=3)} \]
**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**

```r
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)
```
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


See Also

- fixpcmut
- ilmdhc
- adana1
- adana2
- adana3
maxone  

*MAXONE fitness function*

**Description**

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

**Usage**

```r
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**

```r
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**

```r
maxone1(x, ...)
```
maxone2

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
minone

See Also

maxone1, maxone, minone

Examples

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

Description

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

Usage

```r
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

```r
binmat = matrix(nrow=5, ncol=8, byrow=TRUE, c(
1, 0, 1, 0, 1, 1, 0,
1, 1, 1, 0, 1, 0, 0, 0,
1, 1, 1, 0, 1, 0, 0, 0,
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**

```r
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

```r
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

```r
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 **

```r
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)
```

** 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 **

```r
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 **


** 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 **

```r
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)
```
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


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)
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

\[ \text{mx}(x1, x2, cxon, \ldots) \]

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
- **\ldots**: Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References


See Also

cross, px1, kpx, sc, rsc, hux, 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, mp, erx, pbx, pbx2, cx, icx, smc

Examples

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


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

\begin{verbatim}
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)
\end{verbatim}

---

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

\begin{verbatim}
nunimut2(y, lb, ub, g, gmax, mutb, ...)
\end{verbatim}

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

\[
\begin{align*}
\text{lb} &= \text{c}(2, 1, 3, 1, 0, 4) \\
\text{ub} &= \text{c}(10, 15, 8, 5, 6, 9) \\
\text{offspring} &= \text{c}(8, 6, 4, 1, 3, 7) \\
\text{set.seed(12)} \\
\text{nunimut2(offspring, lb=lb, ub=ub, g=1, gmax=100, mutb=0.5)} \\
\text{set.seed(12)} \\
\text{nunimut2(offspring, lb=lb, ub=ub, g=50, gmax=100, mutb=0.5)}
\end{align*}
\]

---

**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**

\[
\text{ox}(x_1, x_2, \text{cxon}, \ldots)
\]

**Arguments**

- \(x_1\): A vector. It contains the chromosomal information of parent-1.
- \(x_2\): A vector. It contains the chromosomal information of parent-2.
- \(\text{cxon}\): Number of offspring to be generated as a result of crossover
- \(\ldots\): Further arguments passed to or from other methods.

**Value**

A matrix containing the generated offsprings.

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**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, pxm, mpmx, upmx, ox2, mpx, erx, pbx, pbx2, cx, icx, smc
Examples

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

---

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

```r
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


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)

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


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)

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


See Also
cross, px1, kpx, sc, rsc, lux, ux, ux2, mx, rrc, disc, atc, cpc, eclc, raoc, dc, ax, hc, sax, wax, lax, bx, ebx, blxa, blxab, lapx, elx, geomx, spherex, pnx, mpmx, upm, ox, oxt, mpx, erx, pb, cx, icx, smc
Examples

```r
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

```r
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**

```r
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**


**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**

```r
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)
```
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)
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)
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, 0, 1, 0, 0, 1)  
px1(parent1, parent2)
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)
**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 $0.1 \times (ub-lb)$.

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

**Usage**

```r
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**

```r
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)
```
**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**

\[
\text{randmut3}(y, \text{lb}, \text{ub}, \text{mutpm}, \ldots)
\]

**Arguments**

- \( y \): A vector. Chromosome of the offspring
- \( \text{lb} \): A vector. Lower bounds of genes
- \( \text{ub} \): A vector. Upper bounds of genes
- \( \text{mutpm} \): Mutation rate
- \( \ldots \): Further arguments passed to or from other methods.

**Value**

- \( \text{mutant} \): A vector. Chromosome of the offspring
- \( \text{mutgen} \): The number of the mutated gene.

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**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

```r
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)
```

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

```r
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


See Also

`mutate, bitmut, randmut, randmut2, randmut3, unimut, unimut2, powmut, powmut2, gaussmut, gaussmut2, gaussmut3, bsearchmut1, bsearchmut2, swapmut, invmut, shufmut, insmut, dismut, invswapmut, insswapmut, invdismut`
Examples

```r
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**

```r
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, mp, err, pbx, pbx2, cx, icx, smc

**Examples**

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

\[ rrc(x_1, x_2, c_{xon}, \ldots) \]

Arguments

- \( x_1 \): A vector. It contains the chromosomal information of parent-1.
- \( x_2 \): A vector. It contains the chromosomal information of parent-2.
- \( c_{xon} \): Number of offspring to be generated as a result of crossover
- \( \ldots \): Further arguments passed to or from other methods.

Value

A matrix containing the generated offsprings.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References


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, mp, erx, pbx, pbx2, cx, icx, smc

Examples

parent1 = c(1, 0, 1, 0, 1, 1, 0)
parent2 = c(1, 1, 0, 1, 0, 0, 1)
rrc(parent1, parent2)
**Description**

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

**Usage**

```r
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**


**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, mpox, px, pbx, pbx2, cx, icx, smc

**Examples**

```r
parent1 = c(1, 0, 1, 0, 1, 1, 1, 0)  
parent2 = c(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.

xon 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, 0.8, 1.1, 1.4, 1.2)

parent2 = c(1.2, 0.8, 0.8, 1.5, 1.2, 1.4)

sax(parent1, parent2)
**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**

```r
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`, `elix`, `geomx`, `spherex`, `pmx`, `mpmx`, `upmx`, `ox`, `ox2`, `mpx`, `erx`, `pbx`, `pbx2`, `cx`, `icx`, `smc`

**Examples**

```r
parent1 = c(1, 0, 1, 0, 1, 1, 0, 1)
parent2 = c(1, 1, 1, 0, 1, 0, 0, 1)
sc(parent1, parent2)
```
**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`, `selrsw`, `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],"
"
"
"
matpool = selboltour(fitvals, selt0=100, selg=95, selgmax=100)
cat(cnames[matpool],"
"
"
"
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

```r
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`, `selrws`, `selsus`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selescale`, `sele`, `selr`, `selr2`

Examples

```r
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 coefficient, \((selb \geq 0.0)\)
selbc     Base of exponent
selc      Scaling parameter
selk      Power factor
sells     Scaling factor
selns     Number of Selection pressure
selps     Percentage of Selection, \((0.0 \leq selps \leq 1.0)\)
sels      Selection pressure, \((1.0 \leq sels \leq 2.0)\)
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


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

```r
# 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))
```

---

**selsers**

*Exponential Ranking Selection*

Description

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

Usage

```r
selsers(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.
selescale

**Value**

The indices of randomly selected individuals are returned.

**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**See Also**

select, selrand, selrswrp, selrws, selrsws, selrss, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selrscale3, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2, sellrs3, sellrs4

**Examples**

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

---

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 coefficient, \((selb \geq 0.0)\)
- `...` 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, selrsws, selrsws, selrss, selrs, selrsw, selrscale, selsscale, selsscale2, sellscale, selrscale, selrsws2, selrws2, sellscale, selrscale2, selrws, sellscale, sellscale, selltour, selltour2

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 I

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, \((1.0 \leq sels \leq 2.0)\)
- **...**: Further arguments passed to or from other methods.

Value

The indices of randomly selected individuals are returned.

Author(s)

Zeynel Cebeci & Erkut Tekeli

References

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

sellrs3

See Also

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

Examples

```r
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")
```

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

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

Arguments

- `fitvals`: Vector of fitness values belonging to individuals
- `ns`: Number of individuals to be selected
- `sels`: Selection pressure, \(1.0 \leq sels \leq 2.0\)
- `...`: 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, selrws, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscale2, selpscale, selescale, seltour, seltour2, selboltour, sellrs, sellrs2
Examples

```r
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**

```r
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**


**See Also**

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

```r
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")
```

Description

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

Usage

```r
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


See Also

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

```r
    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

```r
    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


See Also

- `select`, `selrand`, `selrswrp`, `selrws`, `selrws2`, `selrws`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrwscale`, `selrscale`, `selrscale2`, `selescale`, `seltour`, `seltour2`
**Examples**

```r
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")
```

---

**Description**

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

**Usage**

```r
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`, `selrws`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `sellrscale`, `sellrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`, `selnlrs`, `selers`

**Examples**

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

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`, `selrsrwp`, `selrws`, `selrws2`, `selrsw`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selpscale`, `selescale`, `seltour`, `seltour2`

**Examples**

```r
testvals = c(6.1, 3.5, 6.2, 4.4, 5.2)
cnames = paste0("C",1:length(testvals))
matpool = selrscale(testvals)
cat(cnames[matpool],"n")
```
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, \(1.0 \leq sels \leq 2.0\)
- **...**: 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, selrswp, selrws, selrws2, selrws, selsus, seldet, selwscale, selsscale, selsscale2, selsscale, selscale, selescale, selscale, selsscale, selsus, seldet, selwscale, selsscale, selsscale2, sellscale

Examples

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

```r
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**


**See Also**

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

**Examples**

```r
fitvals = c(6, 1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = selrss(fitvals)
cat("Selected Chromosomes: ", cnames[matpool], "\n")
```
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, selrsc2, selrsc2, selrsc3, selrs, 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 I

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, selrscase, selrscase2, 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")
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

```r
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`, `selrsrwp`, `selrws`, `seltrunc`, `selrss`, `selsus`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`, `selnlrs`, `seler`.

Examples

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


**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")
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, selscale, selrscale, selrsrcale2, 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**

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


See Also

select, selrand, selrswrp, selrws, selrws2, selrws, selsus, seldet, selwscale, selsscale, selsscale2, sellscale, selrscale, selrscalse2, selpscalse, selsscale, 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**


**See Also**

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

**Examples**

```r
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")
```
**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`, `selrws`, `sels`, `seldet`, `selwscale`, `selsscale`, `selsscale2`, `sellscale`, `selrscale`, `selrscale2`, `selpscale`, `selescale`, `seltour`, `seltour2`, `selboltour`, `sellrs`, `sellrs2`, `sellrs3`, `selnrs`, `sels`

**Examples**

```r
fitvals = c(6, -1, 2, 4, 5)
cnames = paste0("C",1:length(fitvals))
matpool = seltrunc(fitvals, selps=0.60)
cat(cnames[matpool],"n")
```
**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, selrsws, selsscale, selsscale2, selrsws2, selrws2, selrsws, selrsws2, selsus, selpsscale, seldet, selrsws2, selrsws, selrsws2, selsus, selpsscale, selrsws2`

Examples

```r
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**

```r
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)
}
```
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

```r
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


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

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

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

**Usage**

```r
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**


**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`, `px2`, `cx`, `icx`

**Examples**

```r
parent1 = c(1, 2, 3, 4, 5, 6, 7, 8)
parent2 = c(4, 6, 7, 3, 2, 1, 8, 6)
smc(parent1, parent2)
```
**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, mp, erx, pbx, pbx2, cx, icx, smc

**Examples**

```r
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)
```
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


Author(s)

Zeynel Cebeci & Erkut Tekeli

References


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**

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

**Arguments**

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

**Value**


**Author(s)**

Zeynel Cebeci & Erkut Tekeli

**References**


**See Also**

- `grdelall`, `elitism`, `grmuplambda`, `grmuplambda2`, `grmuplambda3`, `grmuplambda4`, `grmuvlambda`, `grobin`, `ssrmup1`, `ssrfamtour`, `ssrx`
**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**

**Author(s)**
Zeynel Cebeci & Erkut Tekeli

**See Also**
grdelall, elitism, grmuplambda, grmuplambda2, grmuplambda3, grmuplambda4, grmuvla, grrobin, ssrgenitor, ssrfamtour, 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: Further arguments passed to or from other methods.
### Arguments

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

### Value


### Author(s)

Zeynel Cebeci & Erkut Tekeli

### See Also

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

---

### 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

```r
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


See Also

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

Examples

```r
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

```r
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.
Minimum and maximum difference (range of change)
Similarity percentage of fitness values
Phi convergence
The minimum difference between the last two fitness values
Percentage of difference between the last two best fitness values
System time saved before starting GA
Maximum running time

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

Zeynel Cebeci & Erkut Tekeli

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


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)

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

```r
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


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, bixa, blxab, lapx, elx, geomx, spherex, pmx, mpmx, ox, ox2, mpx, erx, pbx, pbx2, cx, icx, smc

Examples

```r
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

```r
ux(x1, x2, cxon, cxps, ...)
```
ux2

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


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


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


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, px, 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

* evolutionary computing
  findoptima, 45
* genetic algorithms
  findoptima, 45
* global optimum
  findoptima, 45
* graphs
  adana-package, 5
  findoptima, 45
* local optima
  findoptima, 45
* optimization
  adana-package, 5
* optimize
  adana-package, 5

* adaptive genetic algorithms
  adana-package, 5
* attribute
  bin2gray, 17
  bin2int, 18
  int2bin, 73
* binary
  bin2gray, 17
  bin2int, 18
  int2bin, 73
* biologically inspired algorithms
  adana-package, 5
* conversion of numbers
  bin2gray, 17
  bin2int, 18
  int2bin, 73
* evolutionary computing
  adana-package, 5
* hybrid genetic algorithms
  adana-package, 5
* integer
  bin2gray, 17
  bin2int, 18
  int2bin, 73
* manip
  bin2gray, 17
  bin2int, 18
  int2bin, 73
* nature-inspired algorithms
  adana-package, 5
* optimization
  adana-package, 5

adana, 5, 5
adana-package, 5
adana1, 10, 12, 14, 15, 46, 65, 79
adana2, 10, 12, 13, 15, 46, 65, 79
adana3, 10, 12–14, 14, 46, 65, 79
atc, 11, 15, 17, 20, 21, 25, 27, 28, 30, 34, 36,
  37, 39, 43, 50, 58, 63, 64, 76–78, 84,
  85, 87, 90–93, 95, 98, 103–107, 135,
  136, 144–147
ax, 11, 16, 16, 20, 21, 25, 27, 28, 30, 34, 36,
  37, 39, 43, 50, 58, 63, 64, 76–78, 84,
  85, 87, 90–93, 95, 98, 103–107, 135,
  136, 144–147
bestsol, 17
bin2gray, 17, 51, 52
bin2int, 18
bitmut, 11, 19, 22–24, 35, 47–49, 71, 72,
  74–76, 86, 88, 89, 96, 97, 99–102,
  134, 141, 143
blxa, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36,
  37, 39, 43, 50, 58, 63, 64, 76–78, 84,
  85, 87, 90–93, 95, 98, 103–107, 135,
  136, 144–147
blxab, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34,
  36, 37, 39, 43, 50, 58, 63, 64, 76–78,
  84, 85, 87, 90–93, 95, 98, 103–107,
  135, 136, 144–147
boundmut, 11, 19, 22, 23, 24, 35, 47–49, 71,
  72, 74–76, 86, 88, 89, 96, 97,
INDEX

calcM, 26, 41
cpc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
cross, 10, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
cx, 11, 16, 17, 20, 21, 25, 27, 28, 29, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
dc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
decode, 31, 40
decode4int, 32, 41
decodewrit, 33, 42
disc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
dismut, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
ebx, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
eclc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
elitism, 12, 38, 53–58, 137–140
elx, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
encode, 31, 40
encode4int, 26, 32, 41
encodepop, 33, 42
erx, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
evaluate, 10, 11, 44
findoptima, 45
fixpcmut, 12–15, 46, 65, 79
gaussmut, 11, 19, 22–24, 35, 47, 48, 49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
gaussmut2, 11, 19, 22–24, 35, 47, 48, 49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
gaussmut3, 11, 19, 22–24, 35, 47, 48, 49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
geomx, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
gray2bin, 18, 51, 52
gray2bin2, 51, 52
grdelall, 12, 38, 53, 54–58, 137–140
grmulambda, 12, 38, 53, 55–58, 137–140
grmulambda2, 12, 38, 53, 54, 54, 55–58, 137–140
grmulambda3, 12, 38, 53–55, 55, 56–58, 137–140
grmulambda4, 12, 38, 53–55, 56, 57, 58, 137–140
grmulambda5, 12, 38, 53–56, 56, 58, 137–140
grrbin, 12, 38, 53–57, 57, 137–140
hc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
hgaoptim, 12, 59, 61, 62
hgaoptimx, 12, 60, 60, 62
INDEX

powmut, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
powmut2, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
pxl, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147

randmut, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99, 100–102, 134, 141, 143
randmut2, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99, 100, 101, 102, 134, 141, 143
randmut3, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99, 100, 101, 102, 134, 141, 143
raoc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103, 104–107, 135, 136, 144–147
replace, 11
rrc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103, 104, 105–107, 135, 136, 144–147
rsc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103, 104, 105, 106, 107, 135, 136, 144–147
sax, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–105, 106, 107, 135, 136, 144–147
sc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–106, 107, 135, 136, 144–147
seldet, 11, 108, 109, 111–132
select, 10, 11, 108, 109, 110, 112–132

eselers, 11, 111, 111, 119, 123–125, 130, 131
sellrs, 11, 111, 112, 113, 115, 117, 119, 123–125, 130, 131
sellrs2, 11, 111, 112, 114, 115, 117, 119, 123–125, 130, 131
sellrs3, 11, 111, 112, 115, 117, 119, 123–125, 130, 131
selnlrs, 11, 111, 112, 117, 119, 123–125, 130, 131
selrscale, 11, 108, 109, 111–119, 120, 121–132
selrscale2, 11, 108, 109, 111–119, 121, 122–132
selrrs, 11, 108, 109, 111–121, 122, 123–132
selrsws, 11, 108, 109, 111–123, 124, 125–132
selrsws2, 11, 108, 109, 111–124, 125, 126–132
seltour2, 11, 108, 109, 111–129, 130, 131, 132
seltrunc, 11, 111, 119, 123–125, 130, 131
selwscaler, 11, 108, 109, 111–131, 132
show, 83, 133
shufmut, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 143
smc, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
spherex, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–147
ssrfamtour, 12, 38, 53–58, 137, 138–140
ssr-genitor, 12, 38, 53–58, 137, 138, 139, 140
ssr_mup1, 12, 38, 53–58, 137, 138, 139, 140
ssr_x, 12, 38, 53–58, 137–139, 139
swapmut, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 140, 143
terminate, 10, 11, 141
unimut, 11, 19, 22–24, 35, 47–49, 71, 72, 74–76, 86, 88, 89, 96, 97, 99–102, 134, 141, 142
upmx, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 143, 145–147
ux, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144, 146, 147
ux2, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144, 145, 147
wax, 11, 16, 17, 20, 21, 25, 27, 28, 30, 34, 36, 37, 39, 43, 50, 58, 63, 64, 76–78, 84, 85, 87, 90–93, 95, 98, 103–107, 135, 136, 144–146, 146