Package ‘nsga2R’

October 13, 2022

Type Package

Title Elitist Non-Dominated Sorting Genetic Algorithm

Version 1.1

Date 2022-05-21

Author Ching-Shih (Vince) Tsou <cstsou@mail.ntcb.edu.tw>

Maintainer Ming-Chang (Alan) Lee <alan9956@ydu.edu.tw>

Description Box-constrained multiobjective optimization using the elitist non-dominated sorting genetic algorithm - NSGA-II. Fast non-dominated sorting, crowding distance, tournament selection, simulated binary crossover, and polynomial mutation are called in the main program. The methods are described in Deb et al. (2002) <doi:10.1109/4235.996017>.

Depends mco

License LGPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2022-05-23 09:10:06 UTC

R topics documented:

`nsga2R-package` *Elitist Non-dominated Sorting Genetic Algorithm based on R*

Description

Functions for box-constrained multiobjective optimization using the elitist non-dominated sorting genetic algorithm - NSGA-II.

Details

Package:	nsga2R
Type:	Package
Version:	1.0
Date:	2013-06-12
License:	LGPL-3

This package provide functions for box-constrained multiobjective optimization using the elitist non-dominated sorting genetic algorithm - NSGA-II. Fast non-dominated sorting, crowding distance, tournament selection, simulated binary crossover, and polynomial mutation are called in the main program, nsga2R, to complete the search.

Author(s)

Ching-Shih (Vince) Tsou <cstsou@mail.ntcb.edu.tw>

Maintainer: Ming-Chang (Alan) Lee <alan9956@ydu.edu.tw>

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, 6(2), 182-197.

`boundedPolyMutation` *Bounded Polynomial Mutation Operator*

Description

The bounded polynomial mutation operator is a real-parameter genetic operator. Like in the simulated binary crossover operator, the probability distribution is also a polynomial function instead of a normal distribution.

Usage

```
boundedPolyMutation(parent_chromosome, lowerBounds, upperBounds, mprob, mum)
```

Arguments

`parent_chromosome`	Mating pool with decision variables
`lowerBounds`	Lower bounds of each decision variable
`upperBounds`	Upper bounds of each decision variable
`mprob`	Mutation probability
`mum`	Mutation distribution index, it can be any nonnegative real number

Value

Return the offspring population with decision variables

Author(s)

Ching-Shih (Vince) Tsou <`cstsou@mail.ntcb.edu.tw`>

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, **6(2)**, 182-197.

Examples

```
set.seed(1234)
lowerBounds <- rep(0,30)
upperBounds <- rep(1,30)
mprob <- 0.2
MutDistIdx <- 20
matingPool <- matrix(runif(1200, 0, 1), nrow=40, ncol=30)
childAfterM <- boundedPolyMutation(matingPool,lowerBounds,upperBounds,mprob,MutDistIdx)
childAfterM
```

`boundedSBXover` *Bounded Simulated Binary Crossover Operator*

Description

The simulated binary crossover operator is a real-parameter genetic operator. It simulates the working principal of the single-point crossover operator on binary strings.

Usage

```
boundedSBXover(parent_chromosome, lowerBounds, upperBounds, cprob, mu)
```

Arguments

`parent_chromosome`	Mating pool with decision variables
`lowerBounds`	Lower bounds of each decision variable
`upperBounds`	Upper bounds of each decision variable
`cprob`	Crossover probability
`mu`	Crossover distribution index, it can be any nonnegative real number

Value

Return the offspring population with decision variables

Author(s)

Ching-Shih (Vince) Tsou <cstsou@mail.ntcb.edu.tw>

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, **6(2)**, 182-197.

Examples

```
set.seed(1234)
lowerBounds <- rep(0,30)
upperBounds <- rep(1,30)
cprob <- 0.7
XoverDistIdx <- 20
matingPool <- matrix(runif(1200, 0, 1), nrow=40, ncol=30)
childAfterX <- boundedSBXover(matingPool,lowerBounds,upperBounds,cprob,XoverDistIdx)
childAfterX
```

`crowdingDist4frnt` *Crowding Distance Assignment for Each Front*

Description

This function estimates the density of solutions surrounding a particular solution within each front. It calculates the crowding distances of solutions according to their objectives and those within the same front.

Usage

```
crowdingDist4frnt(pop, rnk, rng)
```

Arguments

`pop`	Population matrix including decision variables, objective functions, and non-domination rank
`rnk`	List of solution indices for each front
`rng`	Vector of each objective function range, i.e. the difference between the maximum and minimum objective function value of each objective

Value

Return a matrix of crowding distances of all solutions

Author(s)

Ching-Shih (Vince) Tsou <`cstsou@mail.ntcb.edu.tw`>

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, **6(2)**, 182-197.

See Also

`fastNonDominatedSorting`

Examples

```
library(mco)
popSize <- 50
lowerBounds <- rep(0,30)
upperBounds <- rep(1,30)
varNo <- length(lowerBounds)
objDim <- 2
set.seed(1234)
population <- t(sapply(1:popSize, function(u) array(runif(length(lowerBounds),
  lowerBounds,upperBounds))))
population <- cbind(population, t(apply(population,1,zdt2)))
ranking <- fastNonDominatedSorting(population[,(varNo+1):(varNo+objDim)])
rnkIndex <- integer(popSize)
i <- 1
while (i <= length(ranking)) {
  rnkIndex[ranking[[i]]] <- i
  i <- i + 1
}
population <- cbind(population,rnkIndex)
objRange <- apply(population[,(varNo+1):(varNo+objDim)], 2, max) -
  apply(population[,(varNo+1):(varNo+objDim)], 2, min)
cd <- crowdingDist4frnt(population,ranking,objRange)
cd
```

`fastNonDominatedSorting` *Fast Non-dominated Sorting*

Description

A fast approach to sort non-dominated solutions into different nondomination levels.

Usage

```
fastNonDominatedSorting(inputData)
```

Arguments

`inputData`	Matrix of solutions with objective function values

Value

Return a list of indices for all fronts.

Author(s)

Ching-Shih (Vince) Tsou <cstsou@mail.ntcb.edu.tw>

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, **6(2)**, 182-197.

Examples

```
set.seed(1234)
# randomly generate a polulation of fifty chromosomes, each with two objectives
y <- matrix(runif(100, -5, 5), nrow=50, ncol=2)
rankIdxList <- fastNonDominatedSorting(y)
rankIdxList
```

`nsga2R` *R Based Non-dominated Sorting Genetic Algorithm II*

Description

A fast and elitist multiobjective genetic algorithm based on R.

Usage

```
nsga2R(fn, varNo, objDim, lowerBounds = rep(-Inf, varNo), upperBounds = rep(Inf, varNo),
  popSize = 100, tourSize = 2, generations = 20, cprob = 0.7, XoverDistIdx = 5,
  mprob = 0.2, MuDistIdx = 10)
```

Arguments

`fn`	Objective functions to be minimized
`varNo`	Number of decision variables
`objDim`	Number of objective functions
`lowerBounds`	Lower bounds of each decision variable
`upperBounds`	Upper bounds of each decision variable
`popSize`	Size of population
`tourSize`	Size of tournament
`generations`	Number of generations
`cprob`	Crossover probability
`XoverDistIdx`	Crossover distribution index, it can be any nonnegative real number
`mprob`	Mutation probability
`MuDistIdx`	Mutation distribution index, it can be any nonnegative real number

Value

The returned value is a 'nsga2R' object with the following fields in additional to above NSGA-II settings:

`parameters`	Solutions of decision variables found
`objectives`	Non-dominated objective function values
`paretoFrontRank`	Nondomination ranks (or levels) that each non-dominated solution belongs to
`crowdingDistance`	Crowding distance of each non-dominated solution

Author(s)

Ching-Shih (Vince) Tsou `<cstsou@mail.ntcb.edu.tw>`

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, **6(2)**, 182-197.

Examples

```
# find the non-dominated solutions of zdt3 test problem
results <- nsga2R(fn=zdt3, varNo=30, objDim=2, lowerBounds=rep(0,30), upperBounds=rep(1,30),
  popSize=50, tourSize=2, generations=50, cprob=0.9, XoverDistIdx=20, mprob=0.1,MuDistIdx=3)

plot(results$objectives)
```

`tournamentSelection` *Tournament Selection*

Description

Tournaments are played among several solutions. The best one is chosen according to their non-domination levels and crowding distances. And it is placed in the mating pool.

Usage

```
tournamentSelection(pop, pool_size, tour_size)
```

Arguments

pop	Population matrix with nondomination rank and crowding distance
pool_size	Size of mating pool, usually same as the population size
tour_size	Size of tournament, the selection pressure can be adjusted by varying the tournament size

Value

Return the mating pool with decision variables, objective functions, nondomination level, and crowding distance

Author(s)

Ching-Shih (Vince) Tsou <cstsou@mail.ntcb.edu.tw>

References

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002), " A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, **6(2)**, 182-197.

Examples

```
library(mco)
tourSize <- popSize <- 10
lowerBounds <- rep(0,30)
upperBounds <- rep(1,30)
varNo <- length(lowerBounds)
objDim <- 2
set.seed(1234)
population <- t(sapply(1:popSize, function(u) array(runif(length(lowerBounds),
  lowerBounds,upperBounds))))
population <- cbind(population, t(apply(population,1,zdt3)))
ranking <- fastNonDominatedSorting(population[,(varNo+1):(varNo+objDim)])
rnkIndex <- integer(popSize)
i <- 1
while (i <= length(ranking)) {
  rnkIndex[ranking[[i]]] <- i
  i <- i + 1
}
population <- cbind(population,rnkIndex);
objRange <- apply(population[,(varNo+1):(varNo+objDim)], 2, max) -
  apply(population[,(varNo+1):(varNo+objDim)], 2, min);
cd <- crowdingDist4frnt(population,ranking,objRange)
population <- cbind(population,apply(cd,1,sum))
matingPool <- tournamentSelection(population,popSize,tourSize)
matingPool
```

Index