Package ‘gaoptim’

February 19, 2015

Type Package

Title Genetic Algorithm optimization for real-based and permutation-based problems

Version 1.1

Date 2013-03-24

Author Fernando Tenorio

Maintainer Fernando Tenorio <fnd.tenorio@gmail.com>

Description Performs a Genetic Algorithm Optimization, given a real-based or permutation-based function and the associated search space.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2013-03-25 06:57:57

R topics documented:

gaoptim .................................................. 2
GAPerm .................................................. 2
GAReal .................................................. 7
plot.GAPerm ............................................. 10
plot.GAReal ............................................ 11
print.GAPerm ......................................... 12
print.GAReal ......................................... 12
summary.GAPerm ..................................... 13
summary.GAReal ..................................... 13

Index 14
gaoptim  
*Genetic Algorithm optimization*

**Description**
Function optimization through Genetic Algorithms.

**Details**
Given a real-based or permutation-based function, and the associated search space, `gaoptim` will perform a function maximization using the Genetic Algorithm approach. For better performance, a real-number encoding is used.

All you need to get started is to provide a function and the associated search space - there are sensible defaults to all the other parameters. On the other hand, you can provide custom genetic-operators to control how your population will reproduce and mutate (see the examples).

After setting the algorithm parameters, you can evolve your population and check the results. You don’t need to do this in one step, you can always evolve a small number of generations and query the best solution found. If this solution doesn’t fit your needs, you can keep evolving your population - this approach saves time and computer resources.

**References**

---

gaperm  
*Genetic Algorithm setup*

**Description**
Setup a `GAPerm` object that can be used to perform a permutation-based optimization.

**Usage**
```
GAPerm(FUN, n, popSize = 100, mutRate = 0.1, cxRate = 0.9, eliteRate = 0.4, selection = c("fitness", "uniform"), crossover = c("pmx"), mutation = c("swap"))
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FUN</strong></td>
<td>The fitness function, which should take a vector as argument and return a numeric value (See details).</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>The number of elements to permutate.</td>
</tr>
<tr>
<td><strong>popSize</strong></td>
<td>The population size.</td>
</tr>
<tr>
<td><strong>mutRate</strong></td>
<td>The mutation rate, a numeric value between 0 and 1. When implementing a custom mutation function, this value should be one of the parameters (see details and examples).</td>
</tr>
<tr>
<td><strong>cxRate</strong></td>
<td>The crossover rate, a numeric value between 0 and 1. This parameter specifies the probability of two individuals effectively exchange DNA during crossover. In case the individuals didn’t crossover, the offspring is a exact copy of the parents. When implementing a custom crossover function, this value should be one of the arguments (see details and examples).</td>
</tr>
<tr>
<td><strong>eliteRate</strong></td>
<td>A numeric value between 0 and 1. The eliteRate * popSize best-fitted individuals will automatically be selected for the next generation.</td>
</tr>
<tr>
<td><strong>selection</strong></td>
<td>The selection operator to be used. You can also implement a custom selection function (see details and examples).</td>
</tr>
<tr>
<td><strong>crossover</strong></td>
<td>The crossover operator to be used. You can also implement a custom crossover function (see details and examples).</td>
</tr>
<tr>
<td><strong>mutation</strong></td>
<td>The mutation operator to be used. You can also implement a custom mutation function (see details and examples).</td>
</tr>
</tbody>
</table>

Details

This is the function used to configure and fine-tune a permutation-based optimization. The basic usage requires only the **FUN** parameter (function to be maximized), together with **n** (the number of elements to permutate), all the other parameters have sensible defaults.

The parameters **selection**, **crossover** and **mutation** can also take a custom function as argument, which needs to be in the appropriate format (see the examples). The text below explains the default behaviour for these parameters, which will be useful if you want to override one or more genetic operators.

- **selection**: The fitness option performs a **fitness-proportionate** selection, so that the fittest individuals will have greater chances of being selected. If you choose this option, the value returned by **FUN** (the fitness value) should be **non-negative**. The **uniform** option will randomly sample the individuals to mate, regardless of their fitness value. See the examples if you want to implement a custom selection function.

- **crossover**: The **pmx** option will perform a 'partially mapped crossover' of the individuals DNA. See the references and examples if you need to implement a custom crossover function. The trick with permutation crossover is to make sure that the resulting children are valid permutations.

- **mutation**: The **swap** option will perform a simple swap between specific gene positions, according to the mutation rate specified.
Value

An object of class GAPerm, which you can pass as an argument to plot or summary. This object is a list with the following accessor functions:

- `bestFit`: Returns a vector with the best fitness achieved in each generation.
- `meanFit`: Returns a vector with the mean fitness achieved in each generation.
- `bestIndividual`: Returns a vector with the best solution found.
- `evolve(h)`: This is the function you call to evolve your population. You also need to specify the number of generations to evolve.
- `population`: Returns the current population matrix.

References


Examples

# TSP with 10 cities around a circular pattern
n = 10
R = 10
angs = seq(0, 2*pi, length = n)
xp = R * cos(angs) + rnorm(n)
yp = R * sin(angs) + rnorm(n)
xp = c(xp, xp[1])
yp = c(yp, yp[1])

base.M = matrix(c(xp, yp), ncol = 2)
dist.FUN = function(p)
{
  p = c(p, p[1])
  M.diff = diff(base.M[p, ])
  dists = apply(M.diff, 1, function(x) x[1]^2 + x[2]^2)
  1/sum(dists)
}

gal = GAPerm(dist.FUN, n, popSize = 100, mutRate = 0.3)
gal$evolve(100)
plot(xp, yp, type = 'n', xlab = '', ylab = '')
res = gal$bestIndividual()
res = c(res, res[1])
i = 1:n
xi = base.M[res[i], 1]
yi = base.M[res[i], 2]
xf = base.M[res[i+1], 1]
yf = base.M[res[i+1], 2]

arrows(xi, yi, xf, yf, col = 'red', angle = 10)
text(base.M[res, 1], base.M[res, 2], 1:n, cex = 0.9, col = 'gray20')
# Euro tour problem (See ?optim)
eurodistmat = as.matrix(eurodist)

# This function will be used for the remaining examples
distance = function(sq)
{
  sq = c(sq, sq[1]
  sq2 <- embed(sq, 2)
  1/sum(eurodistmat[cbind(sq2[,2], sq2[,1])])
}

loc = -cmdscale(eurodist, add = TRUE)$points
x = loc[, 1]
y = loc[, 2]
n = nrow(eurodistmat)

set.seed(1)
ga2 = GAPerm(distance, n, popsize = 100, mutRate = 0.3)
ga2$evolve(200)
best = ga2$bestIndividual()
best = c(best, best[1])
best.dist = 1/max(ga2$bestFit())
res = loc[best, ]
i = 1:n

plot(x, y, type = 'n', axes = FALSE, ylab = '', xlab = '')
title ('Euro tour: TSP with 21 cities')
mtext(paste('Best distance found:', best.dist))
arrows(res[i, 1], res[i, 2], res[i + 1, 1], res[i + 1, 2], col = 'red', angle = 10)
text(x, y, labels(eurodist), cex = 0.8, col = 'gray20')

# Euro tour with custom selection
selec.FUN = function(population, fitnessvec, nLeft)
{
  # Chance of being select proportional to fitness sqrt
  idxs = sample(nrow(population), nLeft, prob = sqrt(fitnessvec))

  # Just return the nLeft selected row indexes
  idxs
}

.ga3 = GAPerm(distance, n, mutRate = 0.3, selection = selec.FUN)
.ga3$evolve(200)
best.dist = 1/max(ga3$bestFit())
plot(ga3, main = 'Euro tour: TSP with 21 cities')
mtext(paste('Best distance found:', best.dist))

# Euro tour with custom crossover
# This is the default pmx implementation
crossover.FUN = function(vec1, vec2, prob)
{  
    # prob is the crossover rate
    if (runif(1) > prob) 
        return(matrix(c(vec1, vec2), nrow = 2, byrow = TRUE))
    
    idxs = sample(1:length(vec1), 2)
    vec1.cp = vec1
    
    for (i in idxs) {  
        other.val = vec2[i]
        vec.idx = which(vec1 == other.val)
        vec1[vec.idx] = vec1[i]
        vec1[i] = other.val
    }
    
    for (i in idxs) {
        other.val = vec1.cp[i]
        vec.idx = which(vec2 == other.val)
        vec2[vec.idx] = vec2[i]
        vec2[i] = other.val
    }
    
    matrix(c(vec1, vec2), nrow = 2, byrow = TRUE)
}

ga4 = GAPerm(distance, n, mutRate = 0.3, crossover = crossover.FUN)
ga4$evolve(200)
best.dist = 1/max(ga4$bestFit())
plot(ga4, main = 'Euro tour: TSP with 21 cities')
mtext(paste('Best distance found:', best.dist))

# Euro tour with custom mutation
# This is the default implementation
mutation.FUN = function(M, mutations) {
    
    # M - The population matrix to apply mutation
    # mutations - The number of mutations you supposed to apply, according to mutRate

    rows = sample(1:nrow(M), mutations, replace = FALSE)
cols = t(replicate(mutations, sample(1:n, 2)))
col1 = cols[, 1]
col2 = cols[, 2]
extM1 = matrix(c(rows, col1), ncol = 2)
extM2 = matrix(c(rows, col2), ncol = 2)
tempCol = M[extM1]
M[extM1] = M[extM2]
M[extM2] = tempCol
M}
Description

Setup a GAReal object that can be used to perform a real-based optimization.

Usage

```r
GAReal(FUN, lb, ub, popSize = 100, mutRate = 0.01,
       cxRate = 0.9, eliteRate = 0.4,
       selection = c("fitness", "uniform"),
       crossover = c("blend", "two.points"),
       mutation = c("noise"))
```

Arguments

- **FUN**  
The fitness function, which should take a vector as argument and return a numeric value (See details).
- **lb**  
A numeric vector specifying the lower bounds for the search domain.
- **ub**  
A numeric vector specifying the upper bounds for the search domain.
- **popSize**  
The population size.
- **mutRate**  
The mutation rate, a numeric value between 0 and 1. When implementing a custom mutation function, this value should be one of the parameters (see details and examples).
- **cxRate**  
The crossover rate, a numeric value between 0 and 1. This parameter specifies the probability of two individuals effectively exchange DNA during crossover. In case the individuals didn’t crossover, the offspring is a exact copy of the parents. When implementing a custom crossover function, this value should be one of the arguments (see details and examples).
- **eliteRate**  
A numeric value between 0 and 1. The eliteRate \* popSize best-fitted individuals will automatically be selected for the next generation.
- **selection**  
The selection operator to be used. You can also implement a custom selection function (see details and examples).
- **crossover**  
The crossover operator to be used. You can also implement a custom crossover function (see details and examples).
- **mutation**  
The mutation operator to be used. You can also implement a custom mutation function (see details and examples).
Details

This is the function used to configure and fine-tune a real-based optimization. The basic usage requires only the FUN parameter (function to be maximized), together with the lb and ub parameters (lower and upper search domain), all the other parameters have sensible defaults.

The parameters selection, crossover and mutation can also take a custom function as argument, which needs to be in the appropriate format (see the examples). The text below explains the default behaviour for these parameters, which will be usefull if you want to override one or more genetic operators.

- **selection**: The fitness option performs a fitness-proportionate selection, so that the fittest individuals will have greater chances of being selected. If you choose this option, the value returned by FUN (the fitness value) should be non-negative. The uniform option will randomly sample the individuals to mate, regardless of their fitness value. See the examples if you want to implement a custom selection function.

- **crossover**: The blend option will perform a linear combination of the individuals DNA, effectively introducing new information into the resulting offspring. For details, see Practical genetic algorithms in the references. The two-points option will perform the classic 2-point crossover. See the examples if you need to implement a custom crossover function.

- **mutation**: The default implementation will uniformly sample n mutation points along the population matrix, where n is given by mutRate * popSize * nvars and nvars is the number of variables in your problem. Each sampled locus will be replaced by a random-uniform number between 0 and 1. See the examples to learn how to use a custom mutation function.

Value

An object of class GAREal, which you can pass as an argument to plot or summary. This object is a list with the following accessor functions:

- **bestfit**: Returns a vector with the best fitness achieved in each generation.
- **meanfit**: Returns a vector with the mean fitness achieved in each generation.
- **bestindividual**: Returns a vector with the best solution found.
- **evolve(h)**: This is the function you call to evolve your population.
- **population**: Returns the current population matrix.

References


Examples

# Maximize a trivial 5 variable function
# The function and search-space below will be used for all examples

```r
fitness.FUN = function(x) sum(x)
```
lb = c(0, 0, 0, 0, 0)
ub = c(10, 10, 10, 10, 10)

gal = GAReal(fitness.FUN, lb, ub)
gal$evolve(200)
plot(gal)

# A custom selection example
selec.FUN = function(population, fitnessvec, nleft)
{
  # population - The population matrix
  # fitnessvec - The corresponding fitness vector for the population matrix
  # nleft - The number of individuals you should select
  half = as.integer(nleft/2)
  remain = nleft - half
  idxs = 1:nrow(population)

  # pick half using fitness-proportionate
  rowIdxs = sample(idxs, half, replace = TRUE, prob = fitnessvec)
  # pick the other half randomly
  rowIdxs = c(rowIdxs, sample(idxs, remain, replace = TRUE))

  # Just return the nleft selected row indexes
  return(rowIdxs)
}
ga2 = GAReal(fitness.FUN, lb, ub, selection = selec.FUN)
ga2$evolve(200)
summary(ga2)

# A custom crossover example
crossover.FUN = function(parent1, parent2, prob)
{
  # parent1, parent2 - The individuals to crossover
  # prob - The probability of a crossover happen (cxRate parameter)

  # Respect the cxRate parameter: if DNA is not exchanged, just return the parents
  if (runif(1) > prob)
    return(matrix(c(parent1, parent2), nrow = 2, byrow = TRUE))

  # A simple uniform crossover - just swap the 'genes' with a probability of 0.5
  for (i in 1:length(parent1))
    {
      if (runif(1) > 0.5)
      {
        tempval = parent1[i]
        parent1[i] = parent2[i]
        parent2[i] = tempval
      }
    }
  # You should return a matrix in this format
  return(matrix(c(parent1, parent2), nrow = 2, byrow = TRUE))
A quick way to visualize the GA results.

Usage

```r
## S3 method for class 'GAPerm'
plot(x, xlab = "Generation",
     ylab = "Fitness", main = "GA optimization",
     bestcol = "steelblue", meancol = "tomato", lwd = 2,
     legend.pos = c("bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", ...
```

Arguments

- `x` An object of class GAPerm.
- `xlab` The label for the x-axis.
- `ylab` The label for the y-axis.
- `main` The plot title.
**plot.GAReal**

The color for the best fitness evolution line
meancol The color for the mean fitness evolution line
lwd The line width.
legend.pos The legend position, as a character vector.
... Other parameters (will be ignored).

---

**plot.GAReal**

*Genetic Algorithm plot*

**Description**

A quick way to visualize the GA results.

**Usage**

```r
## S3 method for class 'GAReal'
plot(x, xlab = "Generation",
     ylab = "Fitness", main = "GA optimization",
     bestcol = "steelblue", meancol = "tomato", lwd = 2,
     legend.pos = c("bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", ...
```

**Arguments**

|x| An object of class GAReal. |
|xlab| The label for the x-axis. |
|ylab| The label for the y-axis. |
|main| The plot title. |
|bestcol| The color for the best fitness evolution line |
|meancol| The color for the mean fitness evolution line |
|lwd| The line width. |
|legend.pos| The legend position, as a character vector. |
|...| Other parameters (will be ignored). |

**Examples**

```r
ga = GAReal(function(x) sum(x), rep(0, 5), rep(10, 5))
ga$evolve(200)
plot(ga)
```
print.GAPerm

Print GA results

Description

Prints the GA results.

Usage

```r
## S3 method for class 'GAPerm'
print(x, ...)
```

```r
## S3 method for class 'summaryGAPerm'
print(x, ...)
```

Arguments

- `x` An object of class GAPerm or summaryGAPerm
- `...` Other parameters (will be ignored).

print.GAReal

Print GA results

Description

Prints the GA results.

Usage

```r
## S3 method for class 'GAReal'
print(x, ...)
```

```r
## S3 method for class 'summaryGAReal'
print(x, ...)
```

Arguments

- `x` An object of class GAReal or summaryGAReal
- `...` Other parameters (will be ignored).
**summary.GAPerm**

**Genetic Algorithm summary**

**Description**
A simple summary of the GA results.

**Usage**
```r
## S3 method for class 'GAPerm'
summary(object, ...)
```

**Arguments**
- `object` An object of class GAPerm.
- `...` Other parameters (will be ignored).

**Value**
An object of class summaryGAPerm, which is a list that can be inspected or printed on-screen.

---

**summary.GARes**

**Genetic Algorithm summary**

**Description**
A simple summary of the GA results.

**Usage**
```r
## S3 method for class 'GARes'
summary(object, ...)
```

**Arguments**
- `object` An object of class GARes.
- `...` Other parameters (will be ignored).

**Value**
An object of class summaryGARes, which is a list that can be inspected or printed on-screen.
Index

gaoptim, 2
.gaoptim-package (gaoptim), 2
GAPerm, 2
GAREal, 7

plot.GAPerm, 10
plot.GAREal, 11
print.GAPerm, 12
print.GAREal, 12
print.summaryGAPerm (print.GAPerm), 12
print.summaryGAREal (print.GAREal), 12

summary.GAPerm, 13
summary.GAREal, 13