Package ‘genalg’

March 16, 2015

Version 0.2.0
Date 2015-03-15
Title R Based Genetic Algorithm
Author Egon Willighagen and Michel Ballings
Maintainer Michel Ballings <Michel.Ballings@gmail.com>
Description R based genetic algorithm for binary and floating point chromosomes.

URL https://github.com/egonw/genalg

BugReports https://github.com/egonw/genalg/issues
License GPL-2
Repository CRAN
Date/Publication 2015-03-16 08:45:54
NeedsCompilation no

R topics documented:

plot.rbga .......................................................... 2
rbga ............................................................. 3
rbga.bin .......................................................... 4
summary.rbga ..................................................... 7

Index 9
plot.rbga

R Based Genetic Algorithm Plot Function

Description

Plots features of the genetic algorithm optimization run. The default plot shows the minimal and mean evaluation value, indicating how far the GA has progressed.

The "hist" plot shows for binary chromosome the gene selection frequency, i.e. the times one gene in the chromosome was selected in the current population. In case of floats chromosomes, it will make histograms for each variable to indicate the selected values in the population.

The "vars" plot the evaluation function versus the variable value. This is useful to look at correlations between the variable and the evaluation values.

Usage

```r
## S3 method for class 'rbga'
plot(x, type="default", breaks=10, ...)
```

Arguments

- `x` a rbga object.
- `type` one of "hist", "vars" or "default".
- `breaks` the number of breaks in a histogram.
- `...` options directly passed to the plot function.

Examples

```r
evaluate <- function(string=c()) {
  returnVal = 1 / sum(string);
  returnVal
}

rbga.results = rbga.bin(size=10, mutationChance=0.01, zeroToOneRatio=0.5, evalFunc=evaluate)

plot(rbga.results)
plot(rbga.results, type="hist")
```
**rbga**  
**R Based Genetic Algorithm (floating point chromosome)**

**Description**

A R based genetic algorithm that optimizes, using a user set evaluation function, a set of floats. It takes as input minimum and maximum values for the floats to optimize. The optimum is the chromosome for which the evaluation value is minimal.

It requires a evalFunc method to be supplied that takes as argument the chromosome, a vector of floats. Additionally, the GA optimization can be monitored by setting a monitorFunc that takes a rbga object as argument.

Results can be visualized with `plot.rbga` and summarized with `summary.rbga`.

**Usage**

```r
rbga(stringMin=c(), stringMax=c(), suggestions=NULL, popSize=200, iters=100, mutationChance=NA, elitism=NA, monitorFunc=NULL, evalFunc=NULL, showSettings=FALSE, verbose=FALSE)
```

**Arguments**

- **stringMin**: vector with minimum values for each gene.
- **stringMax**: vector with maximum values for each gene.
- **suggestions**: optional list of suggested chromosomes
- **popSize**: the population size.
- **iters**: the number of iterations.
- **mutationChance**: the chance that a gene in the chromosome mutates. By default $1/(\text{size}+1)$. It affects the convergence rate and the probing of search space: a low chance results in quicker convergence, while a high chance increases the span of the search space.
- **elitism**: the number of chromosomes that are kept into the next generation. By default is about 20% of the population size.
- **monitorFunc**: Method run after each generation to allow monitoring of the optimization
- **evalFunc**: User supplied method to calculate the evaluation function for the given chromosome
- **showSettings**: if true the settings will be printed to screen. By default False.
- **verbose**: if true the algorithm will be more verbose. By default False.
References


See Also

`rbga.bin` `plot.rbga`

Examples

```r
# optimize two values to match pi and sqrt(50)
evaluate <- function(string=c()) {
  returnVal = NA;
  if (length(string) == 2) {
    returnVal = abs(string[1]-pi) + abs(string[2]-sqrt(50));
  } else {
    stop("Expecting a chromosome of length 2!");
  }
  returnVal
}

monitor <- function(obj) {
  # plot the population
  xlim = c(obj$stringMin[1], obj$stringMax[1]);
  ylim = c(obj$stringMin[2], obj$stringMax[2]);
  plot(obj$population, xlim=xlim, ylim=ylim,
       xlab="pi", ylab="sqrt(50)");
}

rbga.results = rbga(c(1, 1), c(5, 10), monitorFunc=monitor,
                     evalFunc=evaluate, verbose=TRUE, mutationChance=0.01)

plot(rbga.results)
plot(rbga.results, type="hist")
plot(rbga.results, type="vars")
```

Description

A R based genetic algorithm that optimizes, using a user set evaluation function, a binary chromosome which can be used for variable selection. The optimum is the chromosome for which the evaluation value is minimal.
It requires a `evalFunc` method to be supplied that takes as argument the binary chromosome, a vector of zeros and ones. Additionally, the GA optimization can be monitored by setting a `monitorFunc` that takes a `rbga` object as argument.

Results can be visualized with `plot.rbga` and summarized with `summary.rbga`.

**Usage**

```r
rbga.bin(size=10,
         suggestions=NULL,
         popSize=200, iters=100,
         mutationChance=NA,
         elitism=NA, zeroToOneRatio=10,
         monitorFunc=NULL, evalFunc=NULL,
         showSettings=FALSE, verbose=FALSE)
```

**Arguments**

- `size` the number of genes in the chromosome.
- `popSize` the population size.
- `iters` the number of iterations.
- `mutationChance` the chance that a gene in the chromosome mutates. By default $1/(\text{size}+1)$. It affects the convergence rate and the probing of search space: a low chance results in quicker convergence, while a high chance increases the span of the search space.
- `elitism` the number of chromosomes that are kept into the next generation. By default is about 20% of the population size.
- `zeroToOneRatio` the chance for a zero for mutations and initialization. This option is used to control the number of set genes in the chromosome. For example, when doing variable selection this parameter should be set high to
- `monitorFunc` Method run after each generation to allow monitoring of the optimization
- `evalFunc` User supplied method to calculate the evaluation function for the given chromosome
- `showSettings` if true the settings will be printed to screen. By default False.
- `verbose` if true the algorithm will be more verbose. By default False.
- `suggestions` optional list of suggested chromosomes

**References**


See Also

`rbga.plot.rbga`

Examples

```r
# a very simplistic optimization
evaluate <- function(string=c()) {
  returnVal = 1 / sum(string);
  returnVal
}

rbga.results = rbga.bin(size=10, mutationChance=0.01, zeroToOneRatio=0.5,
  evalFunc=evaluate)

plot(rbga.results)

# in this example the four variables in the IRIS data
# set are complemented with 36 random variables.
# Variable selection should find the four original
# variables back (example by Ron Wehrens).
## Not run:
data(iris)
library(MASS)
X <- cbind(scale(iris[,1:4]), matrix(rnorm(36*150), 150, 36))
Y <- iris[,5]

iris.evaluate <- function(indices) {
  result = 1
  if (sum(indices) > 2) {
    huhn <- lda(X[,indices==1], Y, CV=TRUE)$posterior
    result = sum(Y != dimnames(huhn)[[2]][apply(huhn, 1,
      function(x) which(x == max(x)))])) / length(Y)
  }
  result
}

monitor <- function(obj) {
  minEval = min(obj$evaluations);
  plot(obj, type="hist");
}

woppa <- rbga.bin(size=40, mutationChance=0.05, zeroToOneRatio=10,
  evalFunc=iris.evaluate, verbose=TRUE, monitorFunc=monitor)

## End(Not run)

# another realistic example: wavelenght selection for PLS on NIR data
## Not run:
library(pls.pcr)
data(NIR)
```
numberOfWavelengths = ncol(NIR$xtrain)
evaluateNIR <- function(chromosome=c()) {
  returnVal = 100
  minLV = 2
  if (sum(chromosome) < minLV) {
    returnVal
  } else {
    xtrain = NIR$xtrain[, chromosome == 1];
    pls.model = pls(xtrain, NIR$ytrain, validation="CV", grpsize=1,
                    ncomp=2:min(10, sum(chromosome)))
    returnVal = pls.model$val$RMS[pls.model$val$nlv-(minLV-1)]
    returnVal
  }
}

monitor <- function(obj) {
  minEval = min(obj$evaluations);
  filter = obj$evaluations == minEval;
  bestObjectCount = sum(rep(1, obj$popSize)[filter]);
  # ok, deal with the situation that more than one object is best
  if (bestObjectCount > 1) {
    bestSolution = obj$population[filter,][1,];
  } else {
    bestSolution = obj$population[filter,];
  }
  outputBest = paste(obj$iter, " #selected=", sum(bestSolution),
                     " Best (Error=", minEval, ";", sep=" ");
  for (var in 1:length(bestSolution)) {
    outputBest = paste(outputBest, bestSolution[var], " ",
                       sep=" ");
  }
  outputBest = paste(outputBest, "\n", sep=" ");
  cat(outputBest);
}
nir.results = rbga.bin(size=numberOfWavelengths, zeroToOneRatio=10,
                       evalFunc=evaluateNIR, monitorFunc=monitor,
                       popSize=200, iters=100, verbose=TRUE)

## End(Not run)
Usage

```ruby
## S3 method for class 'rbga'
summary(object, echo=FALSE, ...)
```

Arguments

- `object`: a rbga object.
- `echo`: if true, the summary will be printed to STDOUT as well as returned.
- `...`: other options (ignored)

Examples

```ruby
evaluate <- function(string=c()) {
  returnVal = 1 / sum(string);
  returnVal
}

rbga.results = rbga.bin(size=10, mutationChance=0.01, zeroToOneRatio=0.5, evalFunc=evaluate)

summary(rbga.results)
```
Index

*Topic multivariate
  plot.rbga, 2
  rbga, 3
  rbga.bin, 4
  summary.rbga, 7

plot.rbga, 2, 3–6
rbga, 3, 6
rbga.bin, 4, 4
summary.rbga, 3, 5, 7