Package 'kofnGA'

November 24, 2015

Type Package

Title A Genetic Algorithm for Fixed-Size Subset Selection

Version 1.2

Date 2015-11-24

Author Mark A. Wolters

Maintainer Mark A. Wolters <mark@mwolters.com>

Description Function kofnGA uses a genetic algorithm to choose a subset of a fixed size k from the integers 1:n, such that a user-supplied objective function is minimized at that subset. The selection step is done by tournament selection based on ranks, and elitism may be used to retain a portion of the best solutions from one generation to the next.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2015-11-24 15:04:02

R topics documented:

`kofnGA-package` *A genetic algorithm for selection of fixed-size subsets*

Description

Genetic algorithm to do subset selection: search for a subset of a fixed size, k, from the integers 1:n, such that user-supplied function is minimized at that subset.

Details

Package:	kofnGA
Type:	Package
Version:	1.1
Date:	2015-02-03
License:	GPL-2

This package provides the function `kofnGA`, which implements a genetic algorithm (GA) to perform subset selection; that is, choosing the best *k* elements from among *n* possibilities. We label the set of possibilities from which we are choosing by the integers `1:n`, and a solution is represented by an index vector, i.e., a vector of integers in the range [1, n] (with no duplicates) indicating which members of the set to choose. The objective function (defining which solution is "best") is arbitrary and user-supplied; the only restriction on this function is that its first argument must be an index vector encoding the solution.

The search results output by `kofnGA` are a list object assigned to the S3 class `GAsearch`. The package includes `summary`, `print`, and `plot` methods for this class to make it easier to inspect the results.

Author(s)

Mark A. Wolters <mark@mwolters.com>

References

Mark A. Wolters (2015), "A Genetic Algorithm for Selection of Fixed-Size Subsets, with Application to Design Problems," *Journal of Statistical Software*, volume 68, Code Snippet 1

Examples

```
# See the examples in the kofnGA help.
```

`kofnGA` *Select the best subset of size k from n alternatives*

Description

Genetic algorithm to do subset selection: search for a subset of a fixed size, k, from the integers 1:n, such that user-supplied function `OF` is minimized at that subset. The selection step is done by tournament selection based on ranks, and elitism may be used to retain the best solutions from one generation to the next.

Usage

```
kofnGA(n, k, OF, popsize = 200, keepbest = floor(popsize/10), ngen = 500,
       tourneysize = max(ceiling(popsize/10), 2), mutprob = 0.01, initpop = NULL,
        verbose=0, ...)
## S3 method for class 'GAsearch'
plot(x, type=c("l","l"), lty=c(1,1), pch=c(-1,-1), col=c("blue", "red"), lwd=c(1,1), ...)
```

```
## S3 method for class 'GAsearch'
summary(object, ...)
## S3 method for class 'GAsearch'
print(x, ...)
```

Arguments

n	The maximum permissible index (i.e., the size of the set we are doing subset selection from). The algorithm chooses a subset of integers from 1 to n.
k	The number of indices to choose (the size of the subset).
OF	The objective function. The first argument of OF should be an index vector of length k containing integers in the range [1, n]. Additional arguments can be passed to OF through ....
popsize	The size of the population; equivalently, the number of offspring produced each generation.
keepbest	This argument is used to implement elitism. The keepbest least fit offspring each generation are replaced by the keepbest most fit members of the previous generation.
ngen	The number of generations to run.
tourneysize	The number of individuals involved in each tournament at the selection stage.
mutprob	The probability of mutation for each of the k chosen indices in each individual. An index chosen for mutation jumps to any other unused index, uniformly at random.
initpop	A popsize-by-k matrix of starting solutions. The final populations from one GA search can be passed as the starting point of the next search. Possibly useful if using this function in an adaptive, iterative, or parallel scheme (see examples).
verbose	An integer controlling the display of progress during search. If verbose takes positive value v, then the iteration number and best objective function value are displayed at the console every v generations. Otherwise nothing is displayed. Default is zero (no display).
object, x	An object of class GAsearch, as returned by kofnGA.
type, lty, pch, col, lwd	Control the appearance of the lines produced by the plot.GAsearch method. Each is a 2-vector: the first element gives the parameter for the plot of average objective function value, and the second element gives the parameter for the plot of the minimum objective function value. See plot or matplot for description and possible values.
...	In kofnga, used to pass other arguments to OF(index_vector,...) as necessary. In plot.GAsearch, used to pass other plot-control arguments. In the summary and print methods, included for consistency with the generic functions.

Details

- Tournament selection involves creating mating "tournaments" where two groups of tourneysize solutions are selected at random without regard to fitness. Within each tournament, victors are

chosen by weighted sampling based on within-tournament fitness ranks (larger ranks given to more fit individuals). The two victors become parents of an offspring. This process is carried out `popsize` times to produce the new population.

- Crossover (reproduction) is carried out by combining the unique elements of both parents and keeping `k` of them, chosen at random.
- Increasing `tourneysize` will put more "selection pressure" on the choice of mating pairs, and will speed up convergence (to a local optimum) accordingly. Smaller `tourneysize` values will conversely promote better searching of the solution space.
- Increasing the size of the elite group (`keepbest`) also promotes more homogeneity in the population, thereby speeding up convergence.

Value

The function returns a list of class "GAsearch" with the following elements:

`bestsol`	A vector of length `k` holding the best solution found.
`bestobj`	The objective function value for the best solution found.
`pop`	A `popsize`-by-`k` matrix holding the final population, row-sorted in order of increasing objective function. Each row is an index vector representing one solution.
`obj`	The objective function values corresponding to each row of pop.
`old`	A list holding information about the search progress. Its elements are:
`old$best`	The sequence of best solutions known over the course of the search (an `(ngen+1)`-by-`k` matrix)
`old$obj`	The sequence of objective function values corresponding to the solutions in `old$best`
`old$avg`	The average population objective function value over the course of the search (a vector of length `ngen+1`). Useful to give a rough indication of population diversity over the search. If the average fitness is close to the best fitness in the population, most individuals are likely quite similar to each other.

Author(s)

Mark A. Wolters <mark@mwolters.com>

References

Mark A. Wolters (2015), "A Genetic Algorithm for Selection of Fixed-Size Subsets, with Application to Design Problems," *Journal of Statistical Software*, volume 68, Code Snippet 1

Examples

```
#---Find the four smallest numbers in a random vector of 100 uniforms---
# Generate the numbers and sort them so the best solution is (1,2,3,4).
Numbers <- sort(runif(100))
Numbers[1:6]                                          #-View the smallest numbers.
ObjFun <- function(v, some_numbers) sum(some_numbers[v])  #-The objective function.
```

```
ObjFun(1:4, Numbers)                                        #-The global minimum.
out <- kofnGA(n = 100, k = 4, OF = ObjFun, ngen = 50, some_numbers = Numbers)  #-Run the GA.
summary(out)
plot(out)


# Note: the following two examples take approximately 30s each to run (on a 2014 laptop)

#---Harder: find the 50x50 principal submatrix of a 500x500 matrix s.t. determinant is max---
# Use eigenvalue decomposition and QR decomposition to make a matrix with known eigenvalues.
n = 500                                          #-Dimension of the matrix.
k = 50                                           #-Size of subset to sample.
eigenvalues = seq(10, 1, length.out=n)           #-Choose the eigenvalues (all positive).
L = diag(eigenvalues)
RandMat = matrix(rnorm(n^2), nrow=n)
Q = qr.Q(qr(RandMat))
M = Q %*% L %*% t(Q)
M = (M+t(M))/2                                  #-Enusre symmetry (fix round-off errors).
ObjFun = function(v,Mat)  -(determinant(Mat[v,v],log=TRUE)$modulus)
out = kofnGA(n=n, k=k, OF=ObjFun, Mat=M)
print(out)
summary(out)
plot(out)

#---For interest: run GA searches iteratively (use initpop argument to pass results)---
# Alternate running with mutation probability 0.05 and 0.005, 50 generations each time.
# Use the same problem as just above (need to run that first).
mutprob = 0.05
result <- kofnGA(n=n, k=k, OF=ObjFun, ngen=50, mutprob=mutprob, Mat=M)  #-First run (random start)
allavg <- result$old$avg                        #-For holding population average OF values
allbest <- result$old$obj                        #-For holding population best OF values
for(i in 2:10) {
  if(mutprob==0.05) mutprob <- 0.005 else mutprob <- 0.05
  result <- kofnGA(n=n, k=k, OF=ObjFun, ngen=50, mutprob=mutprob, initpop=result$pop, Mat=M)
  allavg <- c(allavg, result$old$avg[2:51])
  allbest <- c(allbest, result$old$obj[2:51])
}
plot(0:500, allavg, type="l", col="blue", ylim=c(min(allbest), max(allavg)))
lines(0:500, allbest, col="red")
legend("topright", legend=c("Pop average", "Pop best"), col=c("blue", "red"), bty="n",
       lty=1, cex=0.8)
summary(result)
```

Index