Package ‘ICAFF’

February 19, 2015

Type Package

Title Imperialist Competitive Algorithm

Version 1.0.1

Date 2015-02-06

Depends R (>= 3.0.0), graphics, stats

Author Farimah Houshmand, Farzad Eskandari Ph.D. <Askandari@atu.ac.ir>

Maintainer Farimah Houshmand <hoshmandcomputer@gmail.com>

Description Imperialist Competitive Algorithm (ICA)
<http://en.wikipedia.org/wiki/Imperialist_competitive_algorithm>
is a computational method that is used to solve optimization
problems of different types and it is the mathematical model
and the computer simulation of human social evolution.
The package provides a minimum value for the cost function
and the best value for the optimization variables by
Imperialist Competitive Algorithm.
Users can easily define their own objective function
depending on the problem at hand.
This version has been successfully applied to solve
optimization problems, for continuous functions.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-10 15:56:28

R topics documented:

`ICA` *Finding a minimum value for the optimization variables of a cost function.*

Description

ICA is a function for optimization by Imperialist Competitive Algorithm.

Usage

```
ICA(cost, nvar, ncountries = 80, nimp = 10, maxiter = 100,
           lb = rep(-10, nvar), ub = rep(10, nvar),
           beta = 2, P_revolve = 0.3, zeta = 0.02, ...)
```

Arguments

`cost` A cost function to be minimized. The function accept the parameter values as a numerical vector as its principal argument. Additional arguments may be specified through the `...` argument below.

`nvar` Number of optimization variables of cost function

`ncountries` Number of initial countries

`nimp` Number of Initial Imperialists

`maxiter` Maximum number of iterations allowed.

`lb` Lower limit of the optimization region; a numeric vector of length `nvar`. Will be recycled if necessary.

`ub` Upper limit of the optimization region; a numeric vector of length `nvar`. Will be recycled if necessary.

`beta` Assimilation coefficient.

`P_revolve` Revolution is the process in which the socio-political characteristics of a country change suddenly.

`zeta` Total Cost of Empire = Cost of Imperialist + Zeta * mean(Cost of All Colonies)

`...` Additional arguments, if needed, for the function `cost`.

Details

To use this code, you should only need to prepare your cost function.

Value

An object of class `"ICA"`, a list with components:

`call` The call used.

`postion` The vector of components for the position of the minimum value found.

`value` The minimum value at the optimal position.

`nimp` The remaining number of imperialists at the conclusion of the procedure.

`trace` A 1-column matrix of successive minimum values found at each iteration of the major loop.

`time` The execution time taken to find the best solution.

Note

The steps can be summarized as the below pseudocode:

0) Define objective function or cost function: f(x, ...), x = (x[1], x[2], . . . , x[nvar]) ;

1) Initialization of the algorithm. Generate some random solution in the search space and create initial empires.

2) Assimilation: Colonies move towards imperialist states in different in directions.

3) Revolution: Random changes occur in the characteristics of some countries.

4) Position exchange between a colony and imperialist. A colony with a better position than the imperialist, has the chance to take the control of empire by replacing the existing imperialist.

5) Imperialistic competition: All imperialists compete to take possession of colonies of each other.

6) Eliminate the powerless empires. Weak empires lose their power gradually and they will finally be eliminated.

7) If the stop condition is satisfied, stop, if not go to 2.

8) End

Assimilation: Movement of colonies toward imperialists (Assimilation Policy) Revolution: A sudden change in the socio-political characteristics.

Author(s)

Farimah Houshmand, Farzad Eskandari Ph.D. <Askandari@atu.ac.ir>

Maintainer: Farimah Houshmand <hoshmandcomputer@gmail.com>

References

Atashpaz-Gargari, E. and Lucas, C. (2007). *Imperialist Competitive Algorithm: An algorithm for optimization inspired by imperialistic competition*. IEEE Congress on Evolutionary Computation, Vol. 7, pp. 4661-4666.

Examples

```
## --------cost function: f(x,y) = x * sin(4 * x) + 1.1 * y * sin(2 * y)
## --------search region: -10 <= x, y <= 10

cost <- function(x) {
  x[1] * sin(4 * x[1]) + 1.1 * x[2] * sin(2 * x[2])
}

ICAout <- ICA(cost, nvar = 2, ncountries = 80, nimp = 10,
              maxiter = 100, lb = -10, ub = 10,
                 beta = 2, P_revolve = 0.3, zeta = 0.02)

summary(ICAout)     ## same as the print method
coef(ICAout)        ## get the position of the minimum
cost(coef(ICAout))  ## cost at the minimum
plot(ICAout)        ## show the history of the process
```

plot.ICA *Methods for ICA objects.*

Description

Provide standard methods for manipulating ICA objects, namely printing, plotting, summarising and extracting the position of the minimum.

Usage

```
## S3 method for class 'ICA'
plot(x, ..., xlab = "Iteration", ylab = "Value",
                     main = "ICA History", col = "red")
## S3 method for class 'ICA'
print(x, ...)
## S3 method for class 'ICA'
summary(object, ...)
## S3 method for class 'ICA'
coef(object, ...)
```

Arguments

`x, object` An object of class "ICA".

`xlab, ylab, main, col`
Graphics parameters.

`...` Additional arguments passed on to the method.

Details

Methods for standard generic functions when dealing with objects of class "ICA"

Value

`print` method: the value is printed and returned invisibly.

`summary` method: dummy. Returns the object unchanged.

`plot` method: a plot of the history of the process is produced with a `NULL` return value.

`coef` method: extract the location vector for the minimum value.

Examples

```
## --------cost function: f(x,y) = x * sin(4 * x) + 1.1 * y * sin(2 * y)
## --------search region: -10 <= x, y <= 10

cost <- function(x) {
  x[1] * sin(4 * x[1]) + 1.1 * x[2] * sin(2 * x[2])
}

ICAout <- ICA(cost, nvar = 2, ncountries = 80, nimp = 10,
              maxiter = 100, lb = -10, ub = 10,
              beta = 2, P_revolve = 0.3, zeta = 0.02)

summary(ICAout)     ## same as the print method
coef(ICAout)        ## get the position of the minimum
cost(coef(ICAout))  ## cost at the minimum
plot(ICAout)        ## show the history of the process
```

Index