Package ‘microbats’

February 18, 2016

Type Package
Title An Implementation of Bat Algorithm in R
Version 0.1-1
Date 2016-02-16
Author Seong Hyun Hwang with contributions from Rachel Myoung Moon
Maintainer Seong Hyun Hwang <krshh1412@gmail.com>
Description A nature-inspired metaheuristic algorithm based on the echolocation behavior of micro-bats that uses frequency tuning to optimize problems in both continuous and discrete dimensions. This R package makes it easy to implement the standard bat algorithm on any user-supplied function. The algorithm was first developed by Xin-She Yang in 2010 (<DOI:10.1007/978-3-642-12538-6_6>, <DOI:10.1109/CINTI.2014.7028669>).
Depends R (>= 3.2.1)
License GPL (>= 2)
URL https://github.com/stathwang/microbats
LazyData TRUE
RoxygenNote 5.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2016-02-18 06:51:33

R topics documented:

  bat_optim .................................................. 2

Index 4
Description

The function `bat_optim` implements a nature-inspired metaheuristic algorithm that deals with both continuous and discrete optimization problems. The algorithm is based on the echolocation behavior of microbats and uses frequency tuning.

Usage

`bat_optim(D, NP, N_Gen, A, r, Qmin, Qmax, Lower, Upper, FUN, ...)`

Arguments

- `D` integer: the dimension of the search variables
- `NP` integer: the population size, typically between 10 and 40
- `N_Gen` integer: the number of generations, or iterations
- `A` numeric; loudness, between 0 and 1, can be constant or decreasing
- `r` numeric; pulse rate, must be positive, can be constant or decreasing
- `Qmin` minimum frequency
- `Qmax` maximum frequency
- `Lower` lower bound of the search variables
- `Upper` upper bound of the search variables
- `FUN` the objective function to optimize, must be supplied by a user
- `...` optional arguments to `FUN`

Details

`bat_optim` implements the standard bat algorithm in three robust steps. The first step is to initialize the parameters of algorithm to generate and evaluate the initial population from which to determine the best solution.

Secondly, a population of virtual microbats are moved in a $d$-dimensional search or solution space according to the updating rules of the algorithm: each bat is encoded with a velocity and a location at each iteration in the search space. The location is a solution vector, and the current best solution is achieved.

Then the current best solution is improved using random walks. The new solution is evaluated and updated. See References below for more details.

In essence, frequency tuning acts as mutation to vary the solutions locally; hence, increasing the range of frequencies leads to a global search. The mutation, compared with genetic algorithms, has no crossover but depends on loudness and pulse emission. So technically, varying loudness and pulse emission rates can also make the search intensive approaching the global optimality.

One of the advantages of the bat algorithm is that it can converge very quickly at the initial stage and can switch from exploration to exploitation when the optimality is approaching.
`bat_optim`

**Value**

Returns a list of four values: minimum fitness, population of solutions, fitness, best solution(s)

**Author(s)**

Seong Hyun Hwang, Rachel Myoung Moon

**References**


**Examples**

```r
# find the x-value that gives the minimum of the quadratic function y = x^2 - 3x
# x should then be 1.5
quad_func <- function(x, sol) {
  val = 0
  for (i in 1:length(x)) {
    val <- val + x[i] * x[i] - x[i] * 3
  }
  return(val)
}

# run a simulation using the standard bat algorithm
set.seed(5)  # for reproducive results
fit <- bat_optim(D = 1, NP = 20, N_Gen = 100, A = 0.5, r = 0.5,
                 Qmin = 0, Qmax = 2, Lower = -10, Upper = 10, FUN = quad_func)
x <- fit$best_solution
```
Index

bat_optim, 2