Package: QGA (via r-universe)

September 10, 2024

Type Package

Title Quantum Genetic Algorithm

Version 1.0

Date 2024-05-28

Description Function that implements the Quantum Genetic Algorithm, first proposed by Han and Kim in 2000. This is an R implementation of the 'python' application developed by Lahoz-Beltra (<https://github.com/ResearchCodesHub/QuantumGeneticAlgorithms>). Each optimization problem is represented as a maximization one, where each solution is a sequence of (qu)bits. Following the quantum paradigm, these qubits are in a superposition state: when measuring them, they collapse in a 0 or 1 state. After measurement, the fitness of the solution is calculated as in usual genetic algorithms. The evolution at each iteration is oriented by the application of two quantum gates to the amplitudes of the qubits: (1) a rotation gate (always); (2) a Pauli-X gate (optionally). The rotation is based on the theta angle values: higher values allow a quicker evolution, and lower values avoid local maxima. The Pauli-X gate is equivalent to the classical mutation operator and determines the swap between alfa and beta amplitudes of a given qubit. The package has been developed in such a way as to permit a complete separation between the engine, and the particular problem subject to combinatorial optimization.

License GPL (>= 2)

Encoding UTF-8

LazyLoad yes

Depends R (>= 3.5.0)

Suggests knitr

NeedsCompilation no

URL https://barcaroli.github.io/QGA/, https://github.com/barcaroli/QGA/

BugReports https://github.com/barcaroli/QGA/issues

VignetteBuilder knitr

RoxygenNote 7.3.1

Repository https://barcaroli.r-universe.dev

RemoteUrl https://github.com/barcaroli/qga

RemoteRef HEAD

RemoteSha c66985190159a6aad751a9b092abe0b9e5e8c1a7

Contents

QGA *Quantum Genetic Algorithm*

Description

Main function to execute a Quantum Genetic Algorithm

Usage

```
QGA(
  popsize = 20,
  generation_max = 200,
  nvalues_sol,
  Genome,
  thetainit = 3.1415926535 * 0.05,
  thetaend = 3.1415926535 * 0.025,
  pop_mutation_rate_init = NULL,
  pop_mutation_rate_end = NULL,
  mutation_rate_init = NULL,
  mutation_rate_end = NULL,
  mutation_flag = TRUE,
  plotting = TRUE,
  verbose = TRUE,
  progress = TRUE,
  eval_fitness,
  eval_func_inputs,
  stop_limit = NULL
)
```

Arguments

popsize	the number of generated solutions (population) to be evaluated at each iteration (default is 20)
generation_max	the number of iterations to be performed (default is 200)
nvalues_sol	the number of possible integer values contained in each element (gene) of the solution
Genome	the length of the genome (or chromosome), representing a possible solution
thetainit	the angle (expressed in radiants) to be used when applying the rotation gate when starting the iterations (default is pi * 0.05, where pi = 3.1415926535)
thetaend	the angle (expressed in radiants) to be used when applying the rotation gate at the end of the iterations (default is pi * 0.025, where pi = 3.1415926535)
pop_mutation_rate_init	initial mutation rate to be used when applying the X-Pauli gate, applied to each individual in the population (default is 1/(popsize+1))
pop_mutation_rate_end	final mutation rate to be used when applying the X-Pauli gate, applied to each individual in the population (default is 1/(popsize+1))
mutation_rate_init	initial mutation rate to be used when applying the X-Pauli gate, applied to each element of the chromosome (default is 1/(Genome+1)))
mutation_rate_end	final mutation rate to be used when applying the X-Pauli gate, applied to each element of the chromosome (default is 1/(Genome+1))
mutation_flag	flag indicating if the mutation gate is to be applied or not (default is TRUE)
plotting	flag indicating plotting during iterations
verbose	flag indicating printing fitness during iterations
progress	flag indicating progress bar during iterations
eval_fitness	name of the function that will be used to evaluate the fitness of each solution
eval_func_inputs	specific inputs required by the eval_fitness function
stop_limit	value to stop the iterations if the fitness is higher

Details

This function is the 'engine', which performs the quantum genetic algorithm calling the function for the evaluation of the fitness that is specific for the particulare problem to be optmized.

Value

A numeric vector (positive integers) giving the best solution obtained by the QGA

Examples

```
#----------------------------------------
# Fitness evaluation for Knapsack Problem
#----------------------------------------
KnapsackProblem <- function(solution,
                            eval_func_inputs) {
  solution <- solution - 1
  items <- eval_func_inputs[[1]]
  maxweight <- eval_func_inputs[[2]]
  tot_items <- sum(solution)
  # Penalization
  if (sum(items$weight[solution]) > maxweight) {
    tot_items <- tot_items - (sum(items$weight[solution]) - maxweight)
  }
  return(tot_items)
}
#----------------------------------------
# Prepare data for fitness evaluation
items <- as.data.frame(list(Item = paste0("item",c(1:300)),
                            weight = rep(NA,300)))
set.seed(1234)
items$weight <- rnorm(300,mean=50,sd=20)
hist(items$weight)
sum(items$weight)
maxweight = sum(items$weight) / 2
maxweight
#----------------------
# Perform optimization
popsize = 20
Genome = nrow(items)
solutionQGA <- QGA(popsize = 20,
                generation_max = 500,
                nvalues_sol = 2,
                Genome = nrow(items),
                thetainit = 3.1415926535 * 0.05,
                thetaend = 3.1415926535 * 0.025,
                pop_mutation_rate_init = 1/(popsize + 1),
                pop_mutation_rate_end = 1/(popsize + 1),
                mutation_rate_init = 1,
                mutation_rate_end = 1,
                mutation_flag = TRUE,
                plotting = FALSE,
                verbose = FALSE,
                progress = FALSE,
                eval_fitness = KnapsackProblem,
                eval_func_inputs = list(items,
                                        maxweight))
#----------------------
# Analyze results
solution <- solutionQGA[[1]]
solution <- solution - 1
sum(solution)
```

```
    sum(items$weight[solution])
    maxweight
```

Index