Politechnika Wrocławska

Inteligencja Obliczeniowa i jej zastosowania

Badanie algorytmu genetycznego z zakresu optymalizacji globalnej dla wybranych funkcji testowych

Autorzy:
Paweł Andziul 200648
Marcin Słowiński 200638

Prowadzący:
dr hab. inż. Olgierd Unold,
prof. nadzw. PWr

12 kwietnia 2017

Spis treści

1 Wprowadzenie

Algorytm genetyczny – algorytm heurystyczny, który swoim działaniem przypomina działanie ewolucji w naturze. Osobniki będące zbyt słabe zostają wyeliminowane z populacji w kolejnych pokoleniach, a na ich miejsce przyjmowane są lepsze, silniejsze, bardziej podatne adaptacji. Algorytmy te zakładają możliwość mutacji i krzyżowania wśród potomków, przez co nie zawsze są oni silniejsi od poprzednio wyeliminowanych członków. Dodatkowo wprowadzają pojęcie elity, która jest bezpośrednio przenoszona do następnego - teoretycznie lepszego pokolenia.

dla wybranej funkcji własnej funkcje krzyżowania (dla branina) dla tsp (np–trudny) genetyczny – tsplib wykorzystać do badań (2–3 instancje srednie male duze) z wlasnym operatorem z domyslnym algorytm ga z lokalnym wyszukiwaniem, dla komiwojażera, założyć czy ma lepsze wartości, czy szybciej zbiega, jak operatory się zachowują, psoptim, dla jednej funkcji i komiwojażera

W ramach laboratorium należało przeprowadzić testy algorytmu genetycznego dla różnych parametrów. Jako benchmark oceny należało użyć pakietu „getGlobalOpts" oraz języka R.

Pomiary wykonywano na 2 różnych jednostkach roboczych. Ich parametry nie są istotne z punktu widzenia analizy i możliwości porównania rezultatów.

2 Cel ćwiczeń

Celem jest

3 Opis wykorzystanych funkcji

Gulf, TSP, hybrydowy

Branin jest funkcją z dwoma parametrami. Na ilustracji (rys. 1) przedstawiono jej wykres a poniżej jej wzór (1) [4].

$$f(\boldsymbol{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s \tag{1}$$

, gdzie $x_1 \in [-5, 10]$ oraz $x_2 \in [0, 15]$.

4 Opis własnych funkcji

Opisać jak działa własna funkcja mutacji

5 Wykorzystane narzędzia i biblioteki

5.1 RStudio

5.2 GlobalOptTest

5.3 DoParallel

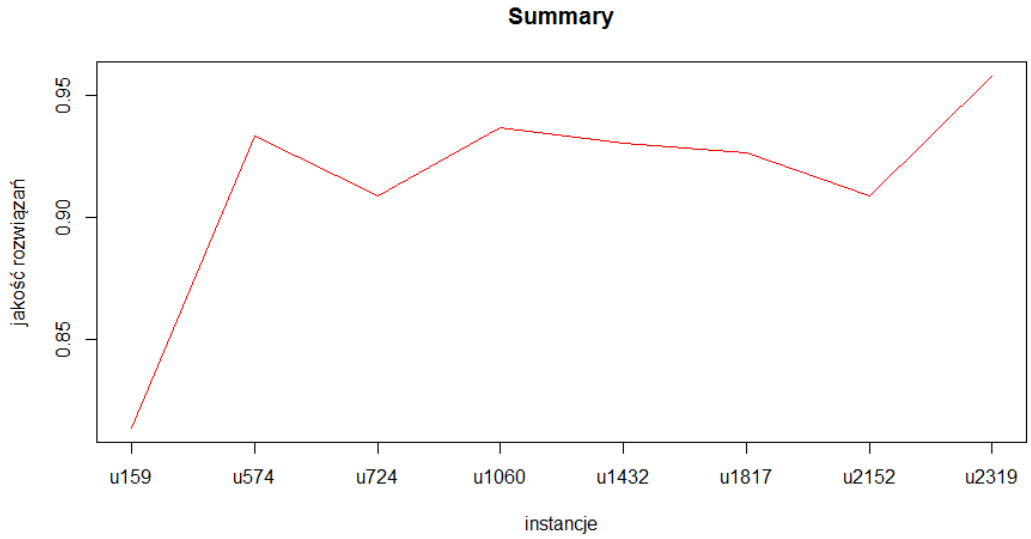
5.4 Rgl

5.5 GA

6 Przebieg badań

Do badań wykorzystano następujące funkcje.

6.1 Gulf



Rysunek 1: Jakość wyników

Z wykresu (rys. 1) wynika, że funkcja ta ma stosunkowo duży obszar w którym może znajdować się minimum oraz dwie strefy w których wartości są dużo większe.

Na kolejnych stronach zamieszczono wyniki pomiarów dla różnych wartości prawdopodobieństwa mutacji dla zmienionej i niezmienionej funkcji mutacji. Pomiary wykonano dla 4 różnych ustawień domyślnych parametrów (serie 1 – 4).

Rysunek 2: Wartość znalezionego minimum funkcji Branin w zależności od prawdopodobieństwa mutacji

6.2 TSP1

6.3 TSP2

6.4 TSP3

6.5 Hybrydowy

7 Podsumowanie

W trakcie prowadzonych badań przetestowano algorytm genetyczny w zadaniu optymalizacji dla 9 funkcji testowych. Analizie poddano wpływ zmiany każdego z parametrów dla 4 różnych konfiguracji pozostałych wartości domyślnych.

Wartość prawdopodobieństwa mutacji i krzyżowania zdaje się odgrywać drugorzędną rolę. Istotne jednak by chociaż jedna z nich była włączona z prawdopodobieństwem większym niż 0.

Najlepszym ustawieniem dla elityzmu jest prawdopodobieństwo rzędu 0,5.

Z pewnością należałoby zwiększyć ilość prób poddawanych uśrednianiu gdyż dla przyjętych 20 wyniki ciągle są niestabilne. Warto by również rozważyć pomijanie kilku najlepszych i najgorszych wyników przed uśrednianiem.

Co ciekawe wyniki są widocznie gorsze przy konfiguracji w której krzyżowanie jest wyłączone a p. mutacji wynosi 0,5. Taka prawidłowość objawia się dla wszystkich badanych funkcji.

8 Implementacja

Poniżej (listing 1) zamieszczono kod napisany w języku R przygotowany w celu umożliwienia przeprowadzenia pomiarów.

Listing 1: Skrypt w języku R wykorzystany do badań

```
# initialize ----
# clean old data
rm(list=ls())
dev.off(dev.list()["RStudioGD"])

# load libraries
require("GA")
require("globalOptTests")
require("rgl")
require("parallel")
require("doParallel")

# custom functions ----
# mutation function
myMutationFunction <- function(object, parent) {
  # get GA population
  population <- parent <- as.vector(object@population[parent, ])

  # calculate randoms
  rnd <- sample(1:length(population), 1)
  rndMinOrMax <- sample(1:2, 1)

  # get min and max from population vector
  max_value <- which.max(population)
  min_value <- which.min(population)

  # if rndMinOrMax is 0 switch random value to min, else switch to max
  if (rndMinOrMax == 0)
  {
    population[rnd] <- min_value;
```

```
  } else
  {
    population[rnd] <- max_value;
  }


  return (population);
}
# crossover function
myCrossoverFunction <- function(object, parent) {

}

# Settings ----

nOfRuns <- 1 # number of runs to calc avg scores

numOfCores <- FALSE # number of cores to use (FALSE, 1 - n)

# colors and titles for plot series
colors <- c("red", "blue", "purple", "black")
series <- c("Seria 1", "Seria 2", "Seria 3", "Seria 4")

# default parameters for measurements
# each row is a different serie
# [mutations,crossovers,populations,iterations,color]
params = matrix(
  c(0, 0, 50, 100, 1,
    0, 0.8, 50, 100, 2,
    0.1, 0, 50, 100, 3,
    0.1, 0.8, 50, 100, 4),
  nrow=4, ncol=5, byrow = TRUE)

# names of functions from globalOptTests package
functions <- c("Branin", "Gulf")#, "CosMix4", "EMichalewicz",
   #"Hartman6", "PriceTransistor", "Schwefel", "Zeldasine20")

# graph settings
graphs <- TRUE #true if you want to print graphs
quality <- 100 #number of probes

# sequences of parameters for each serie
mutationTests <- seq(0, 1, 0.1)
crossoverTests <- seq(0, 1, 0.1)
populationTests <- seq(10, 100, 5)
iterationTests <- seq(10, 200, 10)
elitismTests <- seq(0, 1, 0.1)

# Processing ----

customMeasure <- function(fileName, graphName, values, mType, xlab, main) {

  gMin <- .Machine$integer.max
  gBest <- NA

  # main measurement loop (for each serie and sequence calculate average
```

```
    results)
temp <- c()
for (defRow in 1:nrow(params)) {
  averages <- c()
  for (value in values) {
    sum <- 0
    for (i in 1:nOfRuns) {
      GAmin <- ga(type = "real-valued",
          mutation = myMutationFunction,
          #crossover = myCrossoverFunction,
          fitness = function(xx) -f(xx),
          min = c(B[1,]), max = c(B[2,]),
          popSize = if (mType == "pop") value else params[defRow,3],
          maxiter = if (mType == "itr") value else params[defRow,4],
          pmutation = if (mType == "mut") value else params[defRow,1],
          pcrossover = if (mType == "crs") value else params[defRow,2],
          elitism = if (mType == "elt") value else max(1,
              round(params[defRow,3] * 0.05)),
          parallel = numOfCores)
      solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
      eval <- f(solution[1,])
      if (eval < gMin) {
        gMin <- eval
        gBest <- GAmin
      }
      sum <- sum + eval
    }
    averages <- c(averages, (sum / nOfRuns))
  }
  temp <- c(temp, averages)
}
result <- matrix(c(temp),nrow = nrow(params),ncol = length(values))
write.table(result, file = paste(funcName, fileName, sep=""), row.names=FALSE,
           na="", col.names=FALSE, sep=";")

if (graphs) {

  # save graph with measurement series to file
  png(file = paste(funcName, graphName, ".png", sep=""), width=600,
      height=400, units="px")
  plot(0, 0, main=main,
       ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
       xlim=c(min(values),max(values)),
       type="n", xlab=xlab, ylab="wartosc")
  abline(globalOpt,0, col="green")
  colorNames <- c()
  seriesNames <- c()
  for (i in 1:nrow(params)) {
    color <- colors[params[i,5]]
    colorNames <- c(colorNames, color)
    seriesNames <- c(seriesNames, series[params[i,5]])
    lines(values, result[i,], col = color, type = 'l')
  }
  legend("topright", seriesNames, lwd=rep(2,nrow(params)),
      lty=rep(1,nrow(params)), col=colorNames)
  dev.off()
```

```
139
140     summary(gBest)
141
142     # save overview of best found minimum to file
143     png(file = paste(funcName, graphName, mType, ".png", sep=""), width=600,
            height=400, units="px")
144     filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
145          plot.axes = { axis(1); axis(2);
146            points(solution[1,1], solution[1,2],
147                   pch = 3, cex = 5, col = "black", lwd = 2)
148          }
149     )
150     dev.off()
151
152     # save best fitness graph to file
153     png(file = paste(funcName, graphName, mType, "fitness", ".png", sep=""),
            width=600, height=400, units="px")
154     plot(gBest)
155     dev.off()
156   }
157 }
158
159 for (funcName in functions) {
160
161   # get data from globalOptTests package
162    dim <- getProblemDimen(funcName)
163    B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
164    f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
165                      fnName=funcName, checkDim = TRUE)
166    globalOpt <- getGlobalOpt(funcName)
167
168    if (graphs) {
169      # prepare two versions of graphs (interactive and static)
170      xprobes <- abs(B[2,1] - B[1,1]) / quality
171      yprobes <- abs(B[2,2] - B[1,2]) / quality
172      x <- seq(B[1,1], B[2,1], by = xprobes)
173      y <- seq(B[1,2], B[2,2], by = yprobes)
174      z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
175      nbcol = 100
176      color = rev(rainbow(nbcol, start = 0/6, end = 4/6))
177      zcol  = cut(z, nbcol)
178      persp3d(x, y, z, theta=50, phi=25, expand=0.75, col=color[zcol],
179           ticktype="detailed",axes=TRUE)
180
181      png(file = paste(funcName, "1.png", sep=""), width=600, height=400,
             units="px")
182      persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
183      dev.off()
184    }
185
186    # for each function perform set of measurements
187    customMeasure("resultsMutations.csv", "2", mutationTests, "mut",
188       "p. mutacji", "Znalezione minimum dla roznych prawdopodobienstw mutacji")
189    customMeasure("resultsCrossover.csv", "3", crossoverTests, "crs",
190       "p. krzyzowania", "Znalezione minimum dla roznych prawdopodobienstw
             krzyzowania")
```

```
191     customMeasure("resultsPopulation.csv", "4", populationTests, "pop",
192       "rozmiar populacji", "Znalezione minimum dla roznych rozmiarow populacji")
193     customMeasure("resultsIterations.csv", "5", iterationTests, "itr",
194       "ilosc iteracji", "Znalezione minimum dla roznych ilosci iteracji")
195     customMeasure("resultsElitism.csv", "6", elitismTests, "elt",
196       "elityzm", "Znalezione minimum dla roznych wartosci elityzmu")
197 }
198
199 # whole source code is located here:
200 # https://github.com/cran/GA/tree/master/R
```

Skrypt przygotowano w sposób który umożliwia w pełni automatyczne przeprowadzenie wszystkich pomiarów. Jednocześnie wszystkie wykresy mogą być natychmiast podmienione w sprawozdaniu. Poniżej pokrótce omówiono podstawowe parametry.

- nOfRuns

  Ilość powtórzeń dla każdego pomiaru w celu uśrednienia.

- colors, series

  Wektory kolorów i nazw kolejnych serii pomiarowych.

- params

  Macierz parametrów domyślnych algorytmu dla każdej z serii. W każdym wierszu kolejno są zawarte: p. mutacji, p. krzyżowania, rozmiar populacji, ilość iteracji oraz kolor serii na wykresach.

- functions

  Wektor nazw funkcji dla których przeprowadzane są kolejno pomiary.

Całość informacji niezbędnych do przeprowadzenia obliczeń odczytywana jest na podstawie nazwy funkcji z pakietu „globalOptTests". Są to: rozmiar problemu (ilość parametrów), domyślne ograniczenia, wartość w danym punkcie oraz optimum dla domyślnych ograniczeń.

Literatura

[1] Artur Suchwałko „Wprowadzenie do R dla programistów innych języków” https://cran.r-project.org/doc/contrib/R-dla-programistow-innych-jezykow.pdf

[2] Luca Scrucca „Package GA” https://cran.r-project.org/web/packages/GA/GA.pdf

[3] Surjanovic, S. & Bingham, D. (2013). „Virtual Library of Simulation Experiments: Test Functions and Datasets.” Retrieved April 3, 2017, from http://www.sfu.ca/ ssurjano.

[4] Momin Jamil, Xin-She Yang „A literature survey of benchmark functions for global optimization problems”, Int. Journal of Mathematical Modelling and Numerical Optimisation, Vol. 4, No. 2, pp. 150–194. (2013)

[5] Ajith Abraham, Aboul-Ella Hassanien, Patrick Siarry, Andries Engelbrecht, „Foundations of Computational Intelligence Volume 3” (2009)

[6] Onay Urfalioglu, Orhan Arikan „Self-adaptive randomized and rank-based differential evolution for multimodal problems” (2011)