Politechnika Wrocławska

Inteligencja Obliczeniowa i jej zastosowania

Badanie algorytmu genetycznego z zakresu optymalizacji globalnej dla wybranej funkcji testowej. Przeprowadzenie pomiarów dla algorytmu hybrydowego i optymalizacji rojem cząstek

Autorzy:
Paweł Andziul 200648
Marcin Słowiński 200638

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

26 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. **??**) 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 dla problemu optymalizacji rzeczywistej

Do badań wykorzystano następujące funkcje.

6.1 Gulf

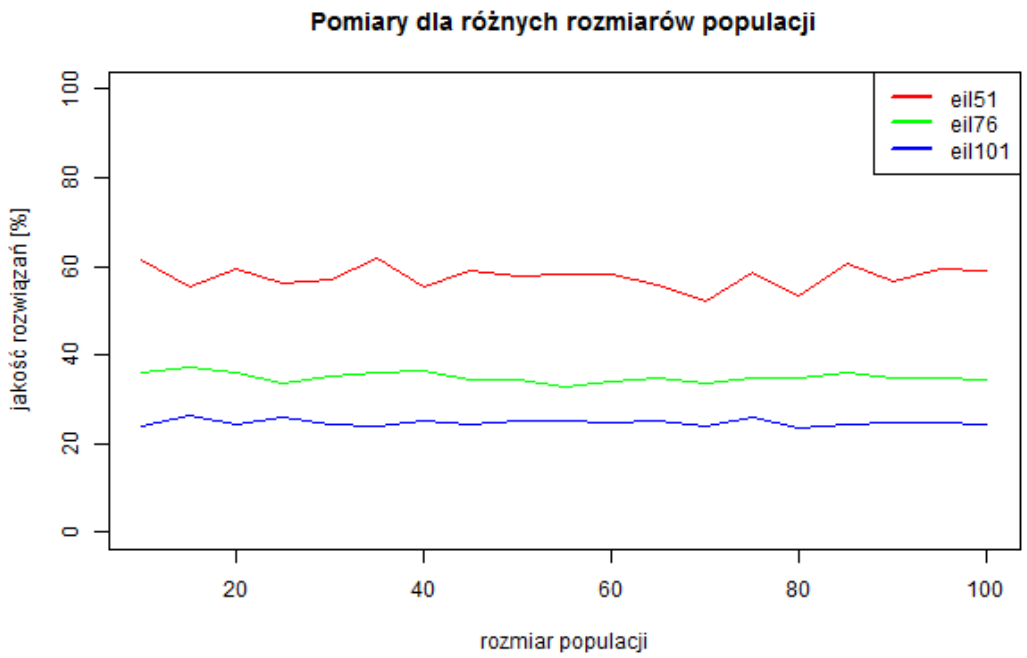
Z wykresu (rys. **??**) wynika, że funkcja ta ma stosunkowo duży obszar w którym może znajdować się minimum oraz dwie strefy w których wartości są dużo większe.

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).

7 Przebieg badań dla problemu komiwojażera

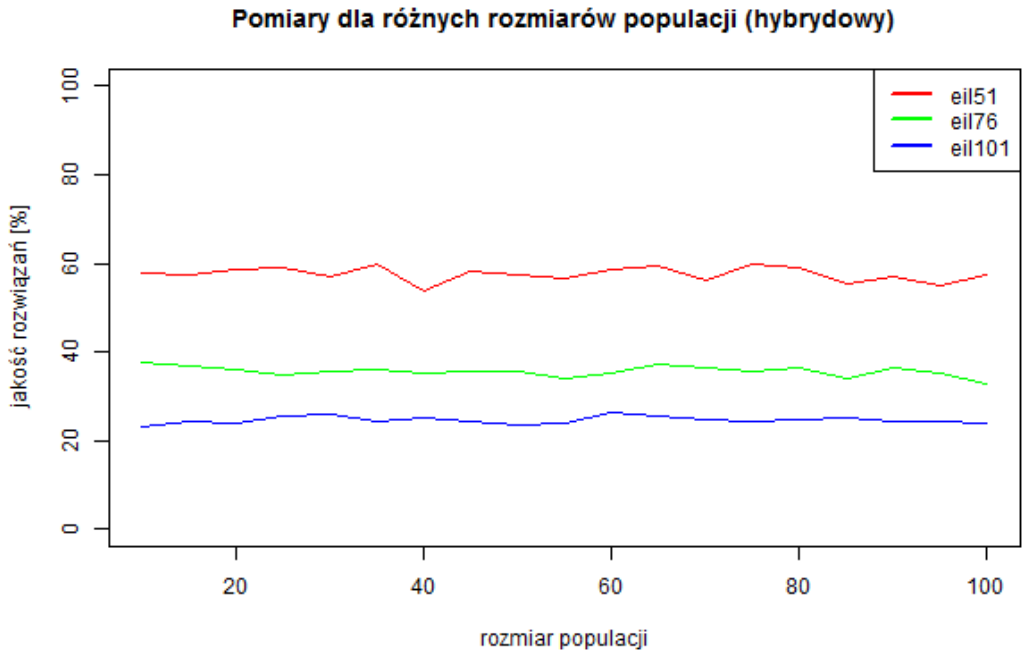
Do badań wykorzystano następujące funkcje.

7.1 Algorytm genetyczny



Rysunek 1: Jakość rozwiązań dla różnych rozmiarów populacji

7.2 Algorytm hybrydowy



Rysunek 2: Jakość rozwiązań dla różnych rozmiarów populacji algorytmu hybrydowego

7.3 Optymalizacja rojem cząstek

8 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.

9 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

```r
# 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(1:10, 1)

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

  # set random element to min value
  population[rnd] = min_value

  return (population);
```

```
}
# crossover function
myCrossoverFunction <- function(object, parent) {

}

# Settings ----

nOfRuns <- 2 # 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)

params = matrix(
  c(0.1, 0.8, 50, 100, 4),
  nrow=1, 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)

# test only mutation (precisely)
mutationTests <- seq(0, 1, 0.01)
crossoverTests <- c(0.2)
populationTests <- c(30)
iterationTests <- c(5)
elitismTests <- c(0.05)

# hybrid algorithm
hybrid = TRUE
poptim = 0.05 #a value [0,1] specifying the probability of performing a local
    search at each iteration of GA (def 0.1)
```

```
86  pressel = 0.5 #a value [0,1] specifying the pressure selection (def 0.5)
87
88  # Processing ----
89
90  customMeasure <- function(fileName, graphName, values, mType, xlab, main) {
91
92    gMin <- .Machine$integer.max
93    gBest <- NA
94
95    # main measurement loop (for each serie and sequence calculate average
          results)
96    temp <- c()
97    for (defRow in 1:nrow(params)) {
98      averages <- c()
99      for (value in values) {
100       sum <- 0
101       for (i in 1:nOfRuns) {
102         GAmin <- ga(type = "real-valued",
103             mutation = myMutationFunction,
104             #crossover = myCrossoverFunction,
105             fitness = function(xx) -f(xx),
106             min = c(B[1,]), max = c(B[2,]),
107             popSize = if (mType == "pop") value else params[defRow,3],
108             maxiter = if (mType == "itr") value else params[defRow,4],
109             pmutation = if (mType == "mut") value else params[defRow,1],
110             pcrossover = if (mType == "crs") value else params[defRow,2],
111             elitism = if (mType == "elt") value else max(1,
                  round(params[defRow,3] * 0.05)),
112             parallel = numOfCores,
113             #hybrid algorithm
114             optim = hybrid,
115             optimArgs = list (
116               poptim = poptim,
117               pressel = pressel))
118         solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
119         eval <- f(solution[1,])
120         if (eval < gMin) {
121           gMin <- eval
122           gBest <- GAmin
123         }
124         sum <- sum + eval
125       }
126       averages <- c(averages, (sum / nOfRuns))
127     }
128     temp <- c(temp, averages)
129   }
130   result <- matrix(c(temp),nrow = nrow(params),ncol = length(values))
131   write.table(result, file = paste(funcName, fileName, sep=""), row.names=FALSE,
132               na="", col.names=FALSE, sep=";")
133
134   if (graphs) {
135
136     # save graph with measurement series to file
137     png(file = paste(funcName, graphName, ".png", sep=""), width=600,
          height=400, units="px")
138     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()

    summary(gBest)

    # save overview of best found minimum to file
    png(file = paste(funcName, graphName, mType, ".png", sep=""), width=600,
        height=400, units="px")
    filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
        plot.axes = { axis(1); axis(2);
          points(solution[1,1], solution[1,2],
                 pch = 3, cex = 5, col = "black", lwd = 2)
        }
    )
    dev.off()

    # save best fitness graph to file
    png(file = paste(funcName, graphName, mType, "fitness", ".png", sep=""),
        width=600, height=400, units="px")
    plot(gBest)
    dev.off()
  }
}

for (funcName in functions) {

  # get data from globalOptTests package
   dim <- getProblemDimen(funcName)
   B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
   f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
                    fnName=funcName, checkDim = TRUE)
   globalOpt <- getGlobalOpt(funcName)

   if (graphs) {
     # prepare two versions of graphs (interactive and static)
     xprobes <- abs(B[2,1] - B[1,1]) / quality
     yprobes <- abs(B[2,2] - B[1,2]) / quality
     x <- seq(B[1,1], B[2,1], by = xprobes)
     y <- seq(B[1,2], B[2,2], by = yprobes)
     z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
     nbcol = 100
     color = rev(rainbow(nbcol, start = 0/6, end = 4/6))
     zcol  = cut(z, nbcol)
```

```
    persp3d(x, y, z, theta=50, phi=25, expand=0.75, col=color[zcol],
          ticktype="detailed",axes=TRUE)

    png(file = paste(funcName, "1.png", sep=""), width=600, height=400,
        units="px")
    persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
    dev.off()
  }

  # for each function perform set of measurements
  customMeasure("resultsMutations.csv", "2", mutationTests, "mut",
    "p. mutacji", "Znalezione minimum dla roznych prawdopodobienstw mutacji")
  customMeasure("resultsCrossover.csv", "3", crossoverTests, "crs",
    "p. krzyzowania", "Znalezione minimum dla roznych prawdopodobienstw
        krzyzowania")
  customMeasure("resultsPopulation.csv", "4", populationTests, "pop",
    "rozmiar populacji", "Znalezione minimum dla roznych rozmiarow populacji")
  customMeasure("resultsIterations.csv", "5", iterationTests, "itr",
    "ilosc iteracji", "Znalezione minimum dla roznych ilosci iteracji")
  customMeasure("resultsElitism.csv", "6", elitismTests, "elt",
    "elityzm", "Znalezione minimum dla roznych wartości elityzmu")
}

# whole source code is located here:
# 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.

Listing 2: Skrypt w języku R wykorzystany do badań dla problemu komiwojażera

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

```
4
5  # load libraries
6  require("GA")
7  require("globalOptTests")
8  require("rgl")
9  require("TSP")
10 require("psoptim")
11
12 numberOfMeasurements <- 2
13
14
15 # TSP with GA ----
16
17 # instances to test and best known solutions
18 instances <- c("eil51", "eil76", "eil101")
19 best_solutions <- c(426, 538, 629)
20 colors <- c("red", "green", "blue")
21
22 tourLength <- function(tour, distMatrix) {
23   tour <- c(tour, tour[1])
24   route <- embed(tour, 2)[,2:1]
25   sum(distMatrix[route])
26 }
27
28 fit <- function(tour, distMatrix) 1/tourLength(tour, distMatrix)
29
30 performTest <- function(testName, graphMain, graphXLab,
31                         sequenceType, sequence,
32                         popsize=50, pcrossover=0.8,
33                         pmutation=0.1, maxiter=100,
34                         optim=FALSE) {
35
36   solution_qualities <- c()
37
38   # each instance as separate serie
39   for (i in 1:length(instances)) {
40
41     fileName = paste("examples/", instances[i], ".tsp", sep="")
42     graphTitle = paste("TSPLIB: ", instances[i], sep="")
43
44     drill <- read_TSPLIB(system.file(fileName, package = "TSP"))
45     D <- as.matrix(dist(drill, method = "euclidean"))
46     N <- max(dim(D))
47
48     solution_quality <- c()
49
50     for (i in 1:length(sequence)) {
51
52       bestTour <- NA
53       bestTourLength <- .Machine$integer.max
54       averageLength <- 0
55
56       for (i in 1:numberOfMeasurements) {
57
58         GA <- ga(type = "permutation",
59                  fitness = fit,
```

```
                distMatrix = D,
                min = 1,
                max = N,
                popSize = if (sequenceType == "popsize") sequence[i] else
                    popsize,
                pcrossover = if (sequenceType == "pcrossover") sequence[i] else
                    pcrossover,
                pmutation = if (sequenceType == "pmutation") sequence[i] else
                    pmutation,
                maxiter = if (sequenceType == "maxiter") sequence[i] else
                    maxiter,
                optim = optim)

      tour <- GA@solution[1, ]
      tl <- tourLength(tour, D)

      if (tl < bestTourLength) {
        bestTourLength <- tl
        bestTour <- tour
      }

      averageLength <- averageLength + (tl - averageLength) / i

    }

    plot(drill, tour, cex=.6, col = "red", pch=3, main = graphTitle)

    solution_quality <- c(solution_quality,
                          (best_solutions[i]/averageLength) * 100)

  }

  solution_qualities <- c(solution_qualities, solution_quality)

}


qualities = matrix(solution_qualities,
              nrow=length(instances), ncol=length(sequence), byrow = TRUE)

# save graph with measurement series to file
png(file = paste(testName, ".png", sep=""), width=600, height=400, units="px")
plot(0, 0, main=graphMain,
     ylim=c(0,100),
     xlim=c(min(sequence),max(sequence)),
     type="n", xlab=graphXLab, ylab="jakość rozwiązań [%]")
for (i in 1:length(instances)) {
  lines(sequence, qualities[i,], col = colors[i], type = 'l')
}
legend("topright", instances, lwd=rep(2,length(instances)),
    lty=rep(1,length(instances)), col=colors)
dev.off()

}

performTest(testName = "tsp_pop",
```

```
            graphMain = "Pomiary dla różnych rozmiarów populacji",
            graphXLab = "rozmiar populacji",
            sequenceType = "popsize", sequence = seq(10, 100, 5))

performTest(testName = "tsp_pop_hyb",
            graphMain = "Pomiary dla różnych rozmiarów populacji (hybrydowy)",
            graphXLab = "rozmiar populacji",
            sequenceType = "popsize", sequence = seq(10, 100, 5), optim = TRUE)

performTest(testName = "tsp_mut",
            graphMain = "Pomiary dla różnych p. mutacji",
            graphXLab = "p. mutacji",
            sequenceType = "pmutation", sequence = seq(0, 1, 0.1))

performTest(testName = "tsp_cross",
            graphMain = "Pomiary dla różnych p. krzyżowania",
            graphXLab = "p. krzyżowania",
            sequenceType = "pcrossover", sequence = seq(0, 1, 0.1))




# PSO tests ----

n <- 500
m.l <- 50
w <- 0.95
c1 <- 0.2
c2 <- 0.2
xmin <- c(-5.12, -5.12)
xmax <- c(5.12, 5.12)
vmax <- c(4, 4)

g <- function(x){
  -(200 + x[,1]^2 + x[,2]^2 + cos(2*pi*x[,2]))
}

psoptim(FUN=g, n=n, max.loop=m.l, w=w, c1=c1, c2=c2,
       xmin=xmin, xmax=xmax, vmax=vmax, seed=5, anim=TRUE)

#determine permutation by number
```

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)