Politechnika Wrocławska

Inteligencja Obliczeniowa i jej zastosowania

Badanie algorytmu genetycznego, memetycznego i rojowego w zadaniu optymalizacji wybranej funkcji testowej oraz badanie algorytmu genetycznego dla problemu TSP

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 Implementacja

Poniżej zamieszczono kody skryptów w języku R przygotowanych w celu umożliwienia przeprowadzenia pomiarów.

Listing 1: Skrypt w języku R wykorzystany do badań optymalizacji funkcji

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

# load libraries
require("GA")
require("globalOptTests")
require("rgl")
require("psoptim")

# 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);
}

# Settings ----

nOfRuns <- 1 # 30 number of runs to calc avg scores

# colors and titles for plot series
colors <- c("red", "purple")
series <- c("GA", "GA + własna mutacja")

# name of function from globalOptTests package
funcName <- "Hartman6"

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

# Processing ----

customGAMeasure <- function(values, mType, xlab, main) {

  # main measurement loop (for each serie and sequence calculate average
      results)
  temp <- c()
  for (serie in 1:length(series)) {
    averages <- c()
    for (value in values) {
      sum <- 0
      for (i in 1:nOfRuns) {

        message(paste("Seria: ", serie))
        message(paste("Sekwencja: ", value))
        message(paste("Przebieg: ", i))

        GAmin <- ga(type = "real-valued",
            mutation = if (serie == 2) myMutationFunction else
                gaControl("real-valued")$mutation,
            fitness = function(xx) -f(xx),
            min = c(B[1,]), max = c(B[2,]),
            popSize = if (mType == "pop") value else 50,
            pmutation = if (mType == "mut") value else 0.1)
        solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
        eval <- f(solution[1,])
        sum <- sum + eval
      }
      averages <- c(averages, (sum / nOfRuns))
    }
    temp <- c(temp, averages)
  }
  result <- matrix(c(temp), nrow = length(series), ncol = length(values))

  if (graphs) {

```

```
    # save graph with measurement series to file
    png(file = paste(funcName, mType, ".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:length(series)) {
      color <- colors[i]
      colorNames <- c(colorNames, color)
      seriesNames <- c(seriesNames, series[i])
      lines(values, result[i,], col = color, type = 'l')
    }
    legend("topright", seriesNames, lwd=rep(2,length(series)),
        lty=rep(1,length(series)), col=colorNames)
    dev.off()

  }
}


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

# perform set of measurements
customGAMeasure(seq(0, 1, 0.1), "mut",
   "p. mutacji", "Znalezione minimum dla różnych p. mutacji")
customGAMeasure(seq(10, 100, 10), "pop",
   "rozmiar populacji", "Znalezione minimum dla różnych rozmiarów populacji")



# hybrid algorithm ----
poptim = 0.05 #a value [0,1] specifying the probability of performing a local
    search at each iteration of GA (def 0.1)
pressel = 0.5 #a value [0,1] specifying the pressure selection (def 0.5)
```

```
133
134 customHybridMeasure <- function(values, mType, xlab, main) {
135
136   averages <- c()
137   for (value in values) {
138     sum <- 0
139     for (i in 1:nOfRuns) {
140
141       message(paste("Sekwencja: ", value))
142       message(paste("Przebieg: ", i))
143
144       GAmin <- ga(type = "real-valued",
145                   fitness = function(xx) -f(xx),
146                   min = c(B[1,]), max = c(B[2,]),
147                   optim = TRUE,
148                   optimArgs = list (
149                     poptim = if (mType == "poptim") value else 0.05,
150                     pressel = if (mType == "pressel") value else 0.5))
151       solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
152       eval <- f(solution[1,])
153       sum <- sum + eval
154     }
155     averages <- c(averages, (sum / nOfRuns))
156   }
157
158   if (graphs) {
159
160     # save graph with measurement series to file
161     png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
          units="px")
162     plot(0, 0, main=main,
163          ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
164          xlim=c(min(values),max(values)),
165          type="n", xlab=xlab, ylab="wartość")
166     abline(globalOpt,0, col="green")
167     lines(values, averages, col = "red", type = 'l')
168     legend("topright", c("memetyczny"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
169     dev.off()
170   }
171 }
172
173 customHybridMeasure(seq(0, 1, 0.05), "poptim",
174                "p. lokalnego searcha", "Znalezione minimum dla różnych poptimów")
175 customHybridMeasure(seq(0, 1, 0.1), "pressel",
176                   "ciśnienie", "Znalezione minimum dla różnych ciśnień")
177
178
179 # PSO tests ----
180
181
182 nOfRuns = 1 # zostaje bo niby nie można uśredniać?
183
184
185 #TODO
186 customPSOMeasure <- function(values, mType, xlab, main) {
187
```

```
  averages <- c()
  for (value in values) {
    sum <- 0
    for (i in 1:nOfRuns) {

      message(paste("Sekwencja: ", value))
      message(paste("Przebieg: ", i))

      GAmin <- ga(type = "real-valued",
                  fitness = function(xx) -f(xx),
                  min = c(B[1,]), max = c(B[2,]),
                  optim = TRUE,
                  optimArgs = list (
                    poptim = if (mType == "poptim") value else 0.05,
                    pressel = if (mType == "pressel") value else 0.5))

      solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
      eval <- f(solution[1,])
      sum <- sum + eval
    }
    averages <- c(averages, (sum / nOfRuns))
  }

  if (graphs) {

    # save graph with measurement series to file
    png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
        units="px")
    plot(0, 0, main=main,
         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
         xlim=c(min(values),max(values)),
         type="n", xlab=xlab, ylab="wartość")
    abline(globalOpt,0, col="green")
    lines(values, averages, col = "red", type = 'l')
    legend("topright", c("memetyczny"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
    dev.off()
  }
}


n <- 500 #ilosc czastek
m.l <- 50 #ilosc przebiegow
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)

#inaczej są parametry podawane, trzeba zrobić dodatkowego wrappera na f()
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=NULL, anim=FALSE)
```

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.

Poniżej skrypt wykorzystany dla problemu komiwojażera.

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

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

# load libraries
require("GA")
require("globalOptTests")
require("rgl")
require("TSP")
require("psoptim")

numberOfMeasurements <- 1 #15

# instances to test and best known solutions
instances <- c("eil51", "eil76", "eil101")
best_solutions <- c(426, 538, 629)
colors <- c("red", "green", "blue")

tourLength <- function(tour, distMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[,2:1]
  sum(distMatrix[route])
}

fit <- function(tour, distMatrix) 1/tourLength(tour, distMatrix)

```

```r
customMutation <- function(object, parent, ...) {

  # Insertion mutation
  parent <- as.vector(object@population[parent,])
  n <- length(parent)
  m <- sample(1:n, size = 1)
  pos <- sample(1:(n-1), size = 1)
  i <- c(setdiff(1:pos,m), m, setdiff((pos+1):n,m))
  mutate <- parent[i]

  # Displacement mutation
  parent <- mutate
  m <- sort(sample(1:n, size = 2))
  m <- seq(m[1], m[2], by = 1)
  l <- max(m)-min(m)+1
  pos <- sample(1:max(1,(n-l)), size = 1)
  i <- c(setdiff(1:n,m)[1:pos], m, setdiff(1:n,m)[-(1:pos)])
  mutate <- parent[na.omit(i)]

  # Scramble mutation
  parent <- mutate
  m <- sort(sample(1:n, size = 2))
  m <- seq(min(m), max(m), by = 1)
  m <- sample(m, replace = FALSE)
  i <- c(setdiff(1:min(m),m), m, setdiff(max(m):n,m))
  mutate <- parent[i]
  return(mutate)

}

performTest <- function(testName, graphMain, graphXLab,
                        sequenceType, sequence,
                        popsize=50, pcrossover=0.8,
                        pmutation=0.1, maxiter=100, mutation = NULL) {

  solution_qualities <- c()

  # each instance as separate serie
  for (i in 1:length(instances)) {

    fileName = paste("examples/", instances[i], ".tsp", sep="")
    graphTitle = paste("TSPLIB: ", instances[i], sep="")

    drill <- read_TSPLIB(system.file(fileName, package = "TSP"))
    D <- as.matrix(dist(drill, method = "euclidean"))
    N <- max(dim(D))

    solution_quality <- c()

    bestTour <- NA
    bestTourLength <- .Machine$integer.max
    averageLength <- 0

    for (s in 1:length(sequence)) {

      for (n in 1:numberOfMeasurements) {
```

```
83
84        message(paste("Instancja: ", i))
85        message(paste("Sekwencja: ", s))
86        message(paste("Pomiar: ", n))
87
88        GA <- ga(type = "permutation",
89                 fitness = fit,
90                 distMatrix = D,
91                 min = 1,
92                 max = N,
93                 popSize = if (sequenceType == "popsize") sequence[s] else
                       popsize,
94                 pcrossover = if (sequenceType == "pcrossover") sequence[s] else
                       pcrossover,
95                 pmutation = if (sequenceType == "pmutation") sequence[s] else
                       pmutation,
96                 maxiter = if (sequenceType == "maxiter") sequence[s] else
                       maxiter,
97                 mutation = if (is.null(mutation))
                       gaControl("permutation")$mutation else mutation)
98
99        tour <- GA@solution[1, ]
100       tl <- tourLength(tour, D)
101
102       if (tl < bestTourLength) {
103         bestTourLength <- tl
104         bestTour <- tour
105       }
106
107       averageLength <- averageLength + (tl - averageLength) / n
108
109     }
110
111     solution_quality <- c(solution_quality,
112                           (best_solutions[i]/averageLength) * 100)
113
114   }
115
116   png(file = paste(testName, "_", instances[i], ".png", sep=""), width=600,
          height=400, units="px")
117   plot(drill, bestTour, cex=.6, col = "red", pch=3, main = graphTitle)
118   dev.off()
119
120   solution_qualities <- c(solution_qualities, solution_quality)
121
122 }
123
124
125 qualities = matrix(solution_qualities,
126                    nrow=length(instances), ncol=length(sequence), byrow = TRUE)
127
128 # save graph with measurement series to file
129 png(file = paste(testName, ".png", sep=""), width=600, height=400, units="px")
130 plot(0, 0, main=graphMain,
131      ylim=c(0,100),
132      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(50, 500, 50))

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

performTest(testName = "tsp_mut_custom",
            graphMain = "Pomiary dla różnych p. mutacji (własny op. mutacji)",
            graphXLab = "p. mutacji",
            sequenceType = "pmutation", sequence = seq(0, 1, 0.1), mutation =
                customMutation)
```

2.1 Opis własnych operatorów

Opisać jak działa własna funkcja mutacji[TODO]

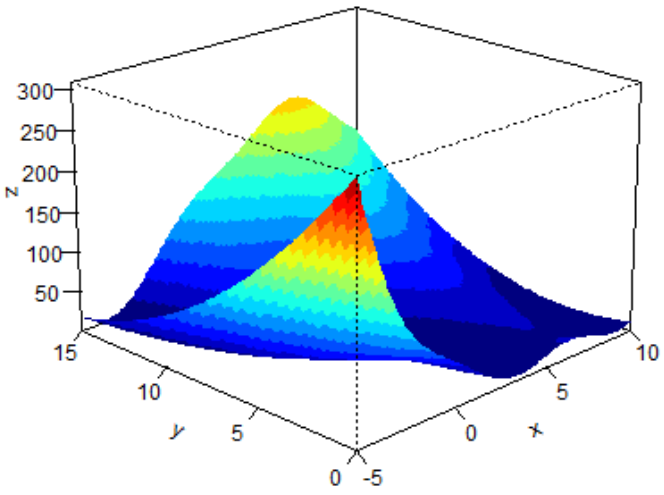
3 Przebieg badań dla problemu optymalizacji rzeczywistej (może jednak Hartman6?)

Badania przeprowadzono dla algorytmu genetycznego w wersji podstawowej oraz hybrydowej a także dla optymalizacji rojem cząstek (PSO).

Funkcja którą poddawano optymalizacji to Branin. 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 $x_2 \in [0, 15]$.



Rysunek 1: Wykres funkcji Branin

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 parametrów kolejnych algorytmów. Pomiary przeprowadzano dla różnych wartości prawdopodobieństwa mutacji i wielkości populacji.

4 Przebieg badań dla problemu komiwojażera

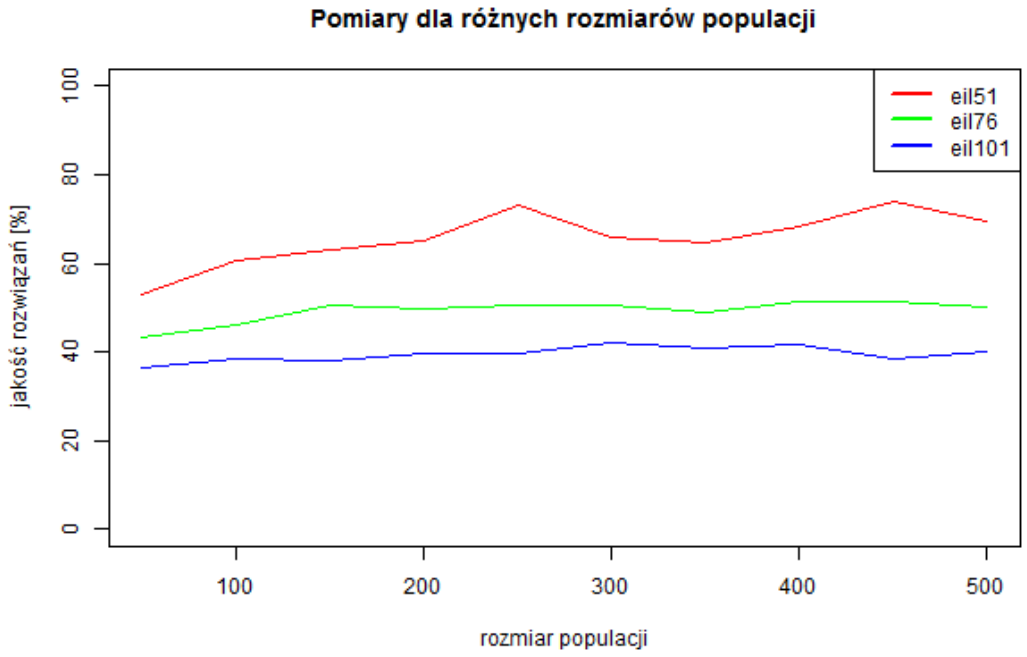
Przeprowadzono badania z zakresu optymalizacji marszruty dla problemu komiwojażera. Wykorzystano trzy instancje problemu z biblioteki TSPLIB:

- eil51
- eil76
- eil101

Jakość rozwiązań wyraża się wzorem:

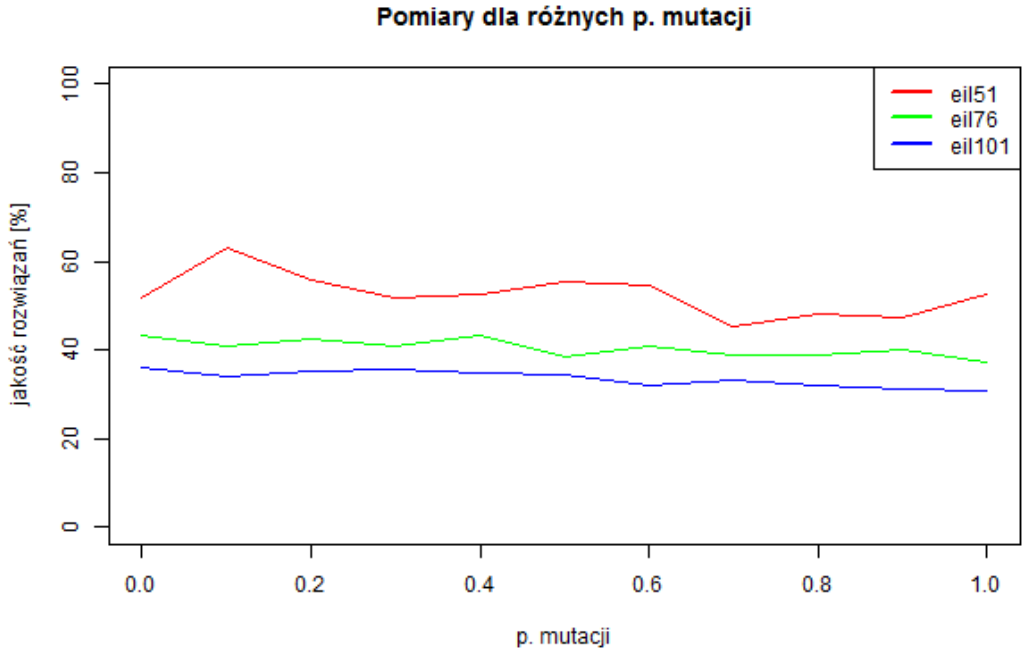
$$quality\ of\ solution = \frac{shortest\ known\ path}{found\ path} * 100\% \tag{2}$$

Na ilustracji (rys. 2) przedstawiono wyniki pomiarów dla różnych rozmiarów populacji.



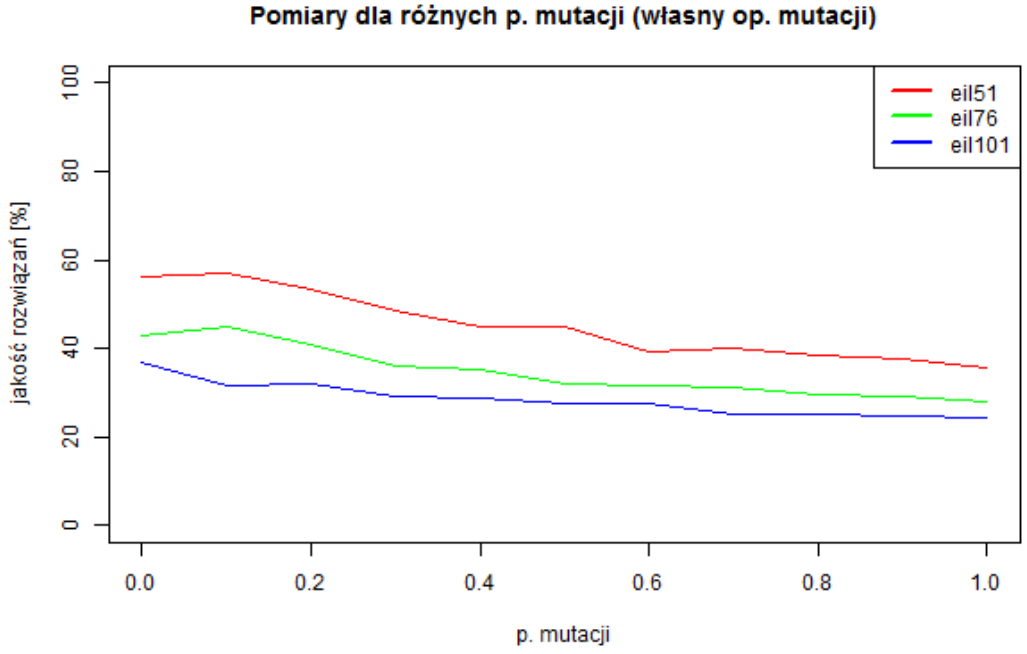
Rysunek 2: Jakość rozwiązań dla różnych rozmiarów populacji

Na ilustracji (rys. 3) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji.



Rysunek 3: Jakość rozwiązań dla różnych wartości p. mutacji

Na ilustracji (rys. 4) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji z niestandardowym operatorem.



Rysunek 4: Jakość rozwiązań dla różnych wartości p. mutacji (dla własnego operatora)

5 Podsumowanie

W trakcie prowadzonych badań przetestowano algorytm genetyczny w zadaniu optymalizacji dla... [TODO]

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)