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

[todo] 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.

Problem komiwojażera - jest zagadnieniem optymalizacyjnym, w którym optymalizowana jest długość trasy między miastami, tak by odnaleźć najkrótszą drogę nie omijając żadnego z nich.

Algorytm memetyczny - hybrydowy algorytm ewolucyjny będący uzupełnieniem algorytmu genetycznego o dodatkowe, poza krzyżowaniem i mutacją, operatory generowania osobników kolejnej generacji.

Algorytm PSO - algorytm optymalizacji rojem cząstek. [todo]

W ramach laboratorium należało przeprowadzić testy algorytmów porównując działanie domyślnej i własnej funkcji mutacji. Dodatkowo należało wykonać testy unikalnych parametrów algorytmu memetycznego i PSO.

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

```r
# 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)

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

  # set random element to min value
```

```
  population[rnd] = min_value

  return (population);
}

# Settings ----

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

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

GAWithHybridSeries <- c("GA", "GA + własna mutacja", "Mem", "Mem + własna
    mutacja")
GAWithHybridColors <- c("red", "purple", "blue", "orange")

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

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

#hybrid algorithm settings
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)

# Processing ----

{
  # 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()
  }
}

#GA
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()

  }
}

customHybridMeasure <- function(values, mType, xlab, main) {
```

```
130
131   averages <- c()
132   for (value in values) {
133     sum <- 0
134     for (i in 1:nOfRuns) {
135
136       message(paste("Sekwencja: ", value))
137       message(paste("Przebieg: ", i))
138
139       GAmin <- ga(type = "real-valued",
140                   fitness = function(xx) -f(xx),
141                   min = c(B[1,]), max = c(B[2,]),
142                   optim = TRUE,
143                   optimArgs = list (
144                     poptim = if (mType == "poptim") value else 0.05,
145                     pressel = if (mType == "pressel") value else 0.5))
146       solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
147       eval <- f(solution[1,])
148       sum <- sum + eval
149     }
150     averages <- c(averages, (sum / nOfRuns))
151   }
152
153   if (graphs) {
154
155     # save graph with measurement series to file
156     png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
          units="px")
157     plot(0, 0, main=main,
158          ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
159          xlim=c(min(values),max(values)),
160          type="n", xlab=xlab, ylab="wartość")
161     abline(globalOpt,0, col="green")
162     lines(values, averages, col = "red", type = 'l')
163     legend("topright", c("memetyczny"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
164     dev.off()
165   }
166 }
167
168 customMeasureGAWithHybrid <- function(values, mType, xlab, main) {
169
170   # main measurement loop (for each serie and sequence calculate average
        results)
171   temp <- c()
172   for (serie in 1:length(GAWithHybridSeries)) {
173     averages <- c()
174     for (value in values) {
175       sum <- 0
176       for (i in 1:nOfRuns) {
177
178         message(paste("Seria: ", GAWithHybridSeries[serie]))
179         message(paste("Sekwencja: ", value))
180         message(paste("Przebieg: ", i))
181
182         if(GAWithHybridSeries[serie] == "GA" || GAWithHybridSeries[serie] == "GA
              + własna funkcja")
```

```
183       {
184         GAmin <- ga(type = "real-valued",
185                     mutation = if (serie == 2) myMutationFunction else
                            gaControl("real-valued")$mutation,
186                     fitness = function(xx) -f(xx),
187                     min = c(B[1,]), max = c(B[2,]),
188                     popSize = if (mType == "pop") value else 50,
189                     pmutation = if (mType == "mut") value else 0.1)
190       }
191       else
192       {
193         GAmin <- ga(type = "real-valued",
194                     mutation = if (serie == 4) myMutationFunction else
                            gaControl("real-valued")$mutation,
195                     fitness = function(xx) -f(xx),
196                     min = c(B[1,]), max = c(B[2,]),
197                     optim = TRUE,
198                     optimArgs = list (
199                       poptim = if (mType == "poptim") value else 0.05,
200                       pressel = if (mType == "pressel") value else 0.5))
201       }
202
203       solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
204       eval <- f(solution[1,])
205       sum <- sum + eval
206     }
207     averages <- c(averages, (sum / nOfRuns))
208   }
209   temp <- c(temp, averages)
210 }
211 result <- matrix(c(temp), nrow = length(GAWithHybridSeries), ncol =
        length(values))
212
213 if (graphs) {
214   # create standalone graph for each serie
215   for (serie in 1:length(GAWithHybridSeries)) {
216     legendColors <- rep("darkslateblue", length(GAWithHybridSeries))
217     legendColors[serie] = "red"
218     # save graph with measurement series to file
219     png(file = paste(funcName, mType, serie, ".png", sep=""), width=600,
            height=400, units="px")
220     plot(0, 0, main=main,
221          ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
222          xlim=c(min(values),max(values)),
223          type="n", xlab=xlab, ylab="wartosc")
224     abline(globalOpt,0, col="green")
225
226     lastLine <- NA
227     seriesNames <- c()
228     for (i in 1:length(GAWithHybridSeries)) {
229       seriesNames <- c(seriesNames, GAWithHybridSeries[i])
230       if (i != serie)
231       {
232         lines(values, result[i,], col = "darkslateblue", type = 'l', lwd = 2)
233       }
234     }
```

```
235       lines(values, result[serie,], col = "red", type = 'l', lwd = 2)
236
237       legend("topright", seriesNames, lwd=rep(2,length(GAWithHybridSeries)),
              lty=rep(1,length(GAWithHybridSeries)), col = legendColors)
238       dev.off()
239     }
240   }
241 }
242
243 #PSO
244
245 customPSOMeasure <- function(values, valueType, xLabel, title) {
246   n <- 500 #ilosc czastek
247   m.l <- 50 #ilosc przebiegow
248   w <- 0.95
249
250   xmin = c(B[1,])
251   xmax = c(B[2,])
252
253   vmax <- c(rep(4, length(xmin)))
254
255   g <- function(x) {
256     vec <- c()
257     for (row in 1:length(x[,1])) {
258       val <- -f(x[row,])
259       vec <- c(vec, val)
260     }
261     vec
262   }
263
264   averages <- c()
265   for (value in values)
266   {
267     sum <- 0
268     for (i in 1:nOfRuns)
269     {
270       message(paste("Sekwencja: ", value))
271       message(paste("Przebieg: ", i))
272
273       result <- psoptim(FUN=g,
274                         n=n,
275                         max.loop=m.l,
276                         w=w,
277                         c1=if (valueType == "c1") value else 0.2,
278                         c2=if (valueType == "c2") value else 0.2,
279                         xmin=xmin,
280                         xmax=xmax,
281                         vmax=vmax,
282                         seed=NULL,
283                         anim=FALSE)
284
285       sum <- sum + f(result$sol)
286
287     }
288     averages <- c(averages, (sum / nOfRuns))
289   }
```

```
290
291   if (graphs) {
292
293     # save graph with measurement series to file
294     png(file = paste(funcName, valueType, ".png", sep=""), width=600,
          height=400, units="px")
295     plot(0, 0, main=title,
296          ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
297          xlim=c(min(values),max(values)),
298          type="n", xlab=xLabel, ylab="wartość")
299     abline(globalOpt,0, col="green")
300     lines(values, averages, col = "red", type = 'l')
301     legend("bottomright", c("PSO"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
302     dev.off()
303   }
304 }
305
306
307
308 # perform set of measurements ----
309 # GA
310 customGAMeasure(seq(0, 1, 0.1), "mut",
311                 "p. mutacji", "Znalezione minimum dla różnych p. mutacji")
312
313 customGAMeasure(seq(10, 100, 10), "pop",
314                 "rozmiar populacji", "Znalezione minimum dla różnych rozmiarów
                      populacji")
315
316 # Hybrid
317 customHybridMeasure(seq(0, 1, 0.01), "poptim",
318                     "p. lokalnego searcha", "Znalezione minimum dla różnych
                          poptimów")
319
320 customHybridMeasure(seq(0, 1, 0.01), "pressel",
321                     "ciśnienie", "Znalezione minimum dla różnych ciśnień")
322
323 # Mixed (GA+Hybrid)
324 customMeasureGAWithHybrid(seq(0, 1, 0.01), "mut",
325                           "p.mutacji", "Znalezione minimum dla różnych p. mutacji")
326
327
328 # PSO
329 customPSOMeasure(seq(0, 1, 0.1), "c1" ,
330                  "c1", "Znalezione minimum dla różnych wartości c1")
331
332 customPSOMeasure(seq(0, 1, 0.01), "c2" ,
333                  "c2", "Znalezione minimum dla różnych wartości c2")
```

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, GAWithHybridSeries, GAWithHybridColors

  Wektory kolorów i nazw kolejnych serii pomiarowych.

- funcName

  Nazwa funkcji dla których przeprowadzane są kolejno pomiary.

- poptim, pressel

  Parametry algorytmu hybrydowego

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.

Dodatkowo warto wspomnieć, iż algorytm „psoptim" w trochę inny sposób niż genetyczny przekazuje parametry do ewaluowanej funkcji. W tym przypadku jest to macierz w której kolumny to kolejne parametry natomiast wiersze odpowiadają kolejnym cząstkom. Zatem wymagane jest by funkcja umożliwiała wektorową ewaluację parametrów. Z uwagi na wykorzystywanie funkcji z pakietu „globalOptTests" wymagało to utworzenia odpowiedniego wrappera.

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)

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) {

        message(paste("Instancja: ", i))
        message(paste("Sekwencja: ", s))
        message(paste("Pomiar: ", n))

```

```
      GA <- ga(type = "permutation",
              fitness = fit,
              distMatrix = D,
              min = 1,
              max = N,
              popSize = if (sequenceType == "popsize") sequence[s] else
                  popsize,
              pcrossover = if (sequenceType == "pcrossover") sequence[s] else
                  pcrossover,
              pmutation = if (sequenceType == "pmutation") sequence[s] else
                  pmutation,
              maxiter = if (sequenceType == "maxiter") sequence[s] else
                  maxiter,
              mutation = if (is.null(mutation))
                  gaControl("permutation")$mutation else mutation)

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

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

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

    }

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

  }

  png(file = paste(testName, "_", instances[i], ".png", sep=""), width=600,
      height=400, units="px")
  plot(drill, bestTour, cex=.6, col = "red", pch=3, main = graphTitle)
  dev.off()

  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(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)
```

Poniżej pokrótce omówiono podstawowe parametry.

- numberOfMeasurements

  Ilość powtórzeń dla każdego pomiaru w celu uśrednienia.

- instances

  Instancje problemu TSP, dla których przeprowadzane są testy.

- best_solutions

  Wektor zawierający optymalne rozwiązania dla instancji.

- colors

  Kolory dla wykresu.

2.1 Opis własnych operatorów

Własna funkcja mutacji została utworzona w taki sposób by nie doprowadzić do sytuacji, w której przekroczona zostanie minimalna lub maksymalna wartość populacji.

Jej działanie opiera się na wybraniu minimalnej jednostki z populacji i podmianie innej, losowej na znaleziona minimalną. Gwarantuje to niepojawienie się z populacji wartości nieoczekiwanej, lecz tylko te otrzymane podczas działania algorytmu.

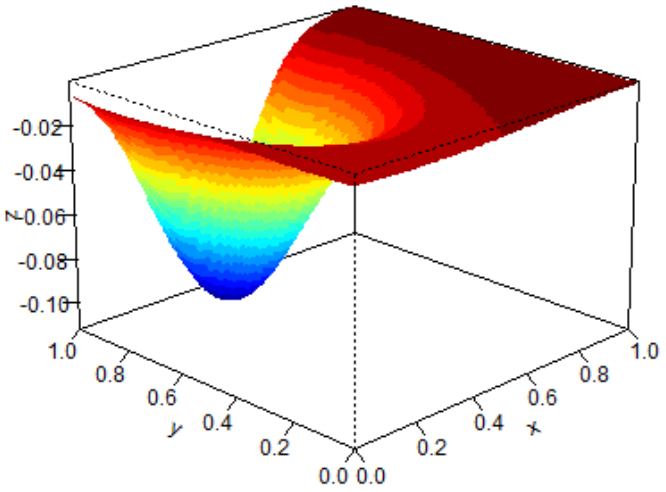
3 Przebieg badań dla problemu optymalizacji rzeczywistej Hartman6

Badania przeprowadzono dla algorytmu genetycznego w wersji podstawowej, ze zmienioną funkcją mutacji oraz hybrydowej, a także dla optymalizacji rojem cząstek (PSO).

Hartman6 jest funkcją określoną dla ilości parametrów równej 6. Na ilustracji (rys. 1) przedstawiono jej wykres dla pierwszych dwóch. Poniżej zamieszczono jej wzór (1).

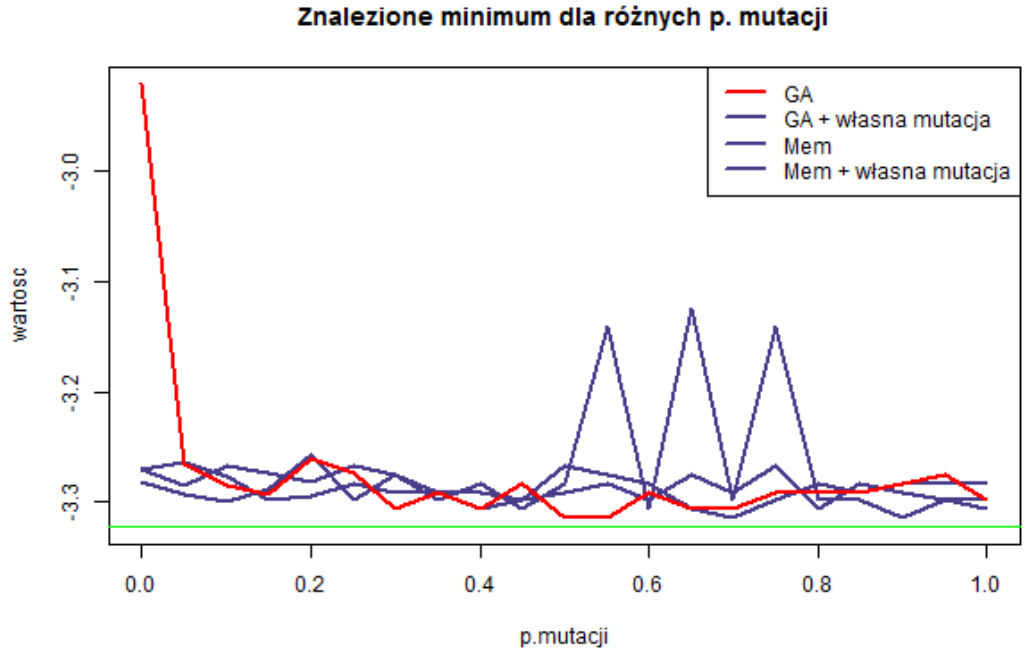
$$f(\boldsymbol{x}) = -\sum_{i=1}^{4} c_i \exp[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2] \tag{1}$$

, gdzie $x_i \in [0,1]$, $i \in \{1,...,6\}$.

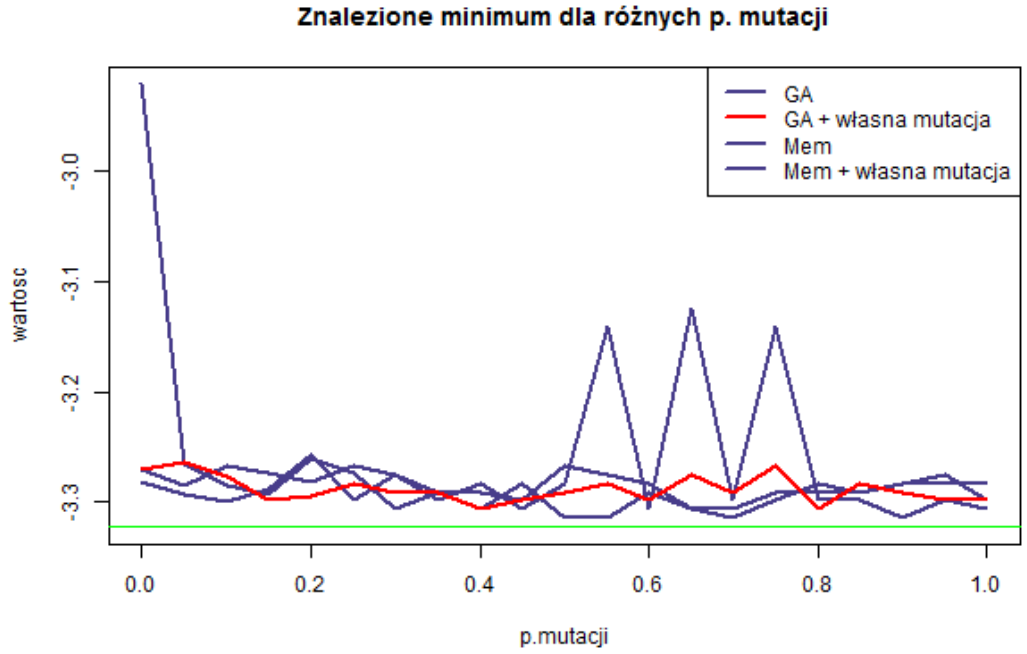


Rysunek 1: Wykres funkcji Hartman6

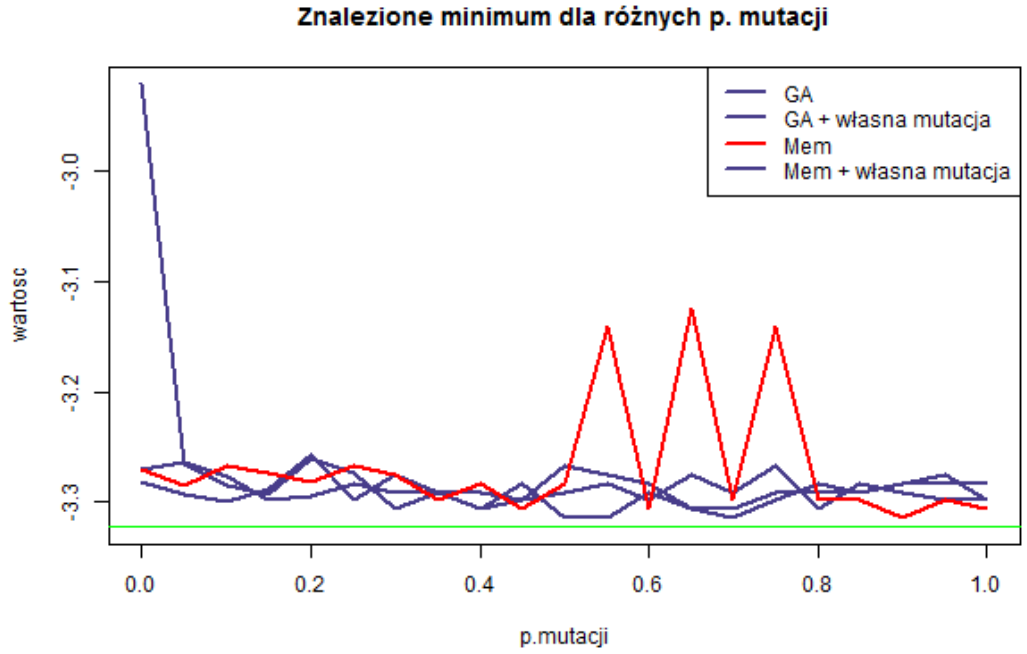
Na kolejnych stronach zamieszczono wyniki badań porównawczych mutacji przeprowadzonych na algorytmie.



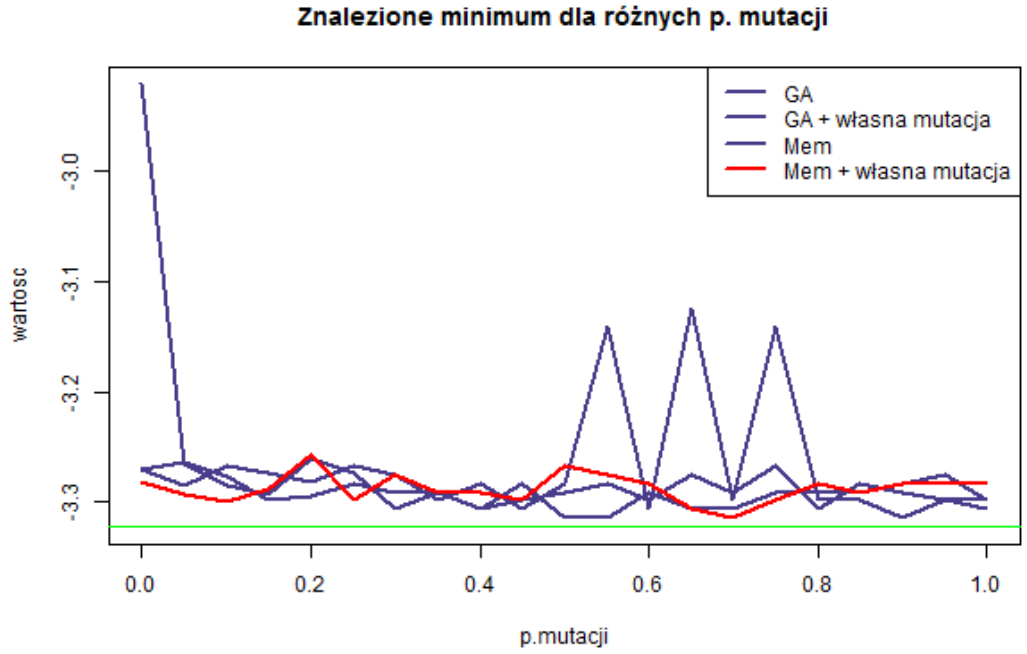
Rysunek 2: Wykres funkcji Hartman6



Rysunek 3: Wykres funkcji Hartman6



Rysunek 4: Wykres funkcji Hartman6



Rysunek 5: Wykres funkcji Hartman6

Z wykresów badań można odczytać zbliżone wyniki dla różnych funkcji mutacji.

Żadna z funkcji nie osiągnęła minimum co świadczy o tym, że sama mutacja do tego nie wystarczy.

Zauważalny jest również niski wpływ własnej funkcji mutacji na otrzymywane wyniki. Pod względem jakości rozwiązań nie odstaje ona od istniejących implementacji.

Na wykresach można zauważyć znaczące pogorszenie się wyników dla funkcji memetycznej z domyślną funkcją mutacji. W przedziale 0.5-0.8 wygenerowała ona wyniki oddalone od średniej pozostałych.

4 Przebieg badań dla problemu komiwojażera

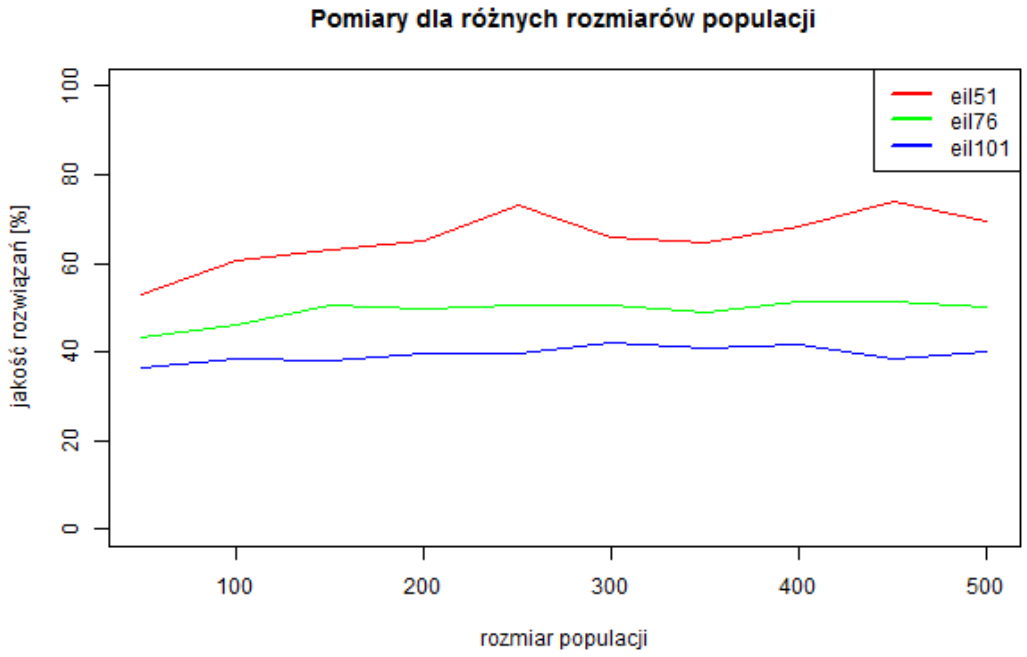
Przeprowadzono badania z zakresu optymalizacji marszruty dla problemu komiwojażera. Wykorzystano trzy instancje problemu z biblioteki TSPLIB:

- eil51
- eil76
- eil101

Jakość rozwiązań wyraża się wzorem:

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

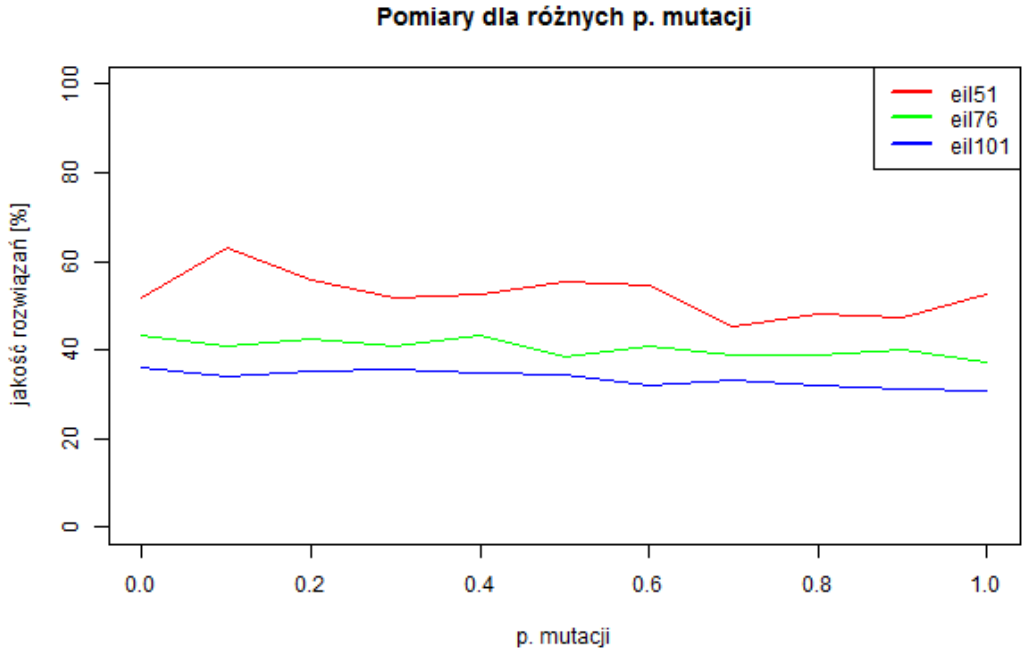
Na ilustracji (rys. 6) przedstawiono wyniki pomiarów dla różnych rozmiarów populacji.



Rysunek 6: Jakość rozwiązań dla różnych rozmiarów populacji

Dla wszystkich badanych instancji zwiększenie populacji wpłynęło pozytywnie na jakość otrzymanego rozwiązania, co najbardziej obrazuje instancja "eli51", która poprawiła się o ok. 20 punktów procentowych w badanym obszarze.

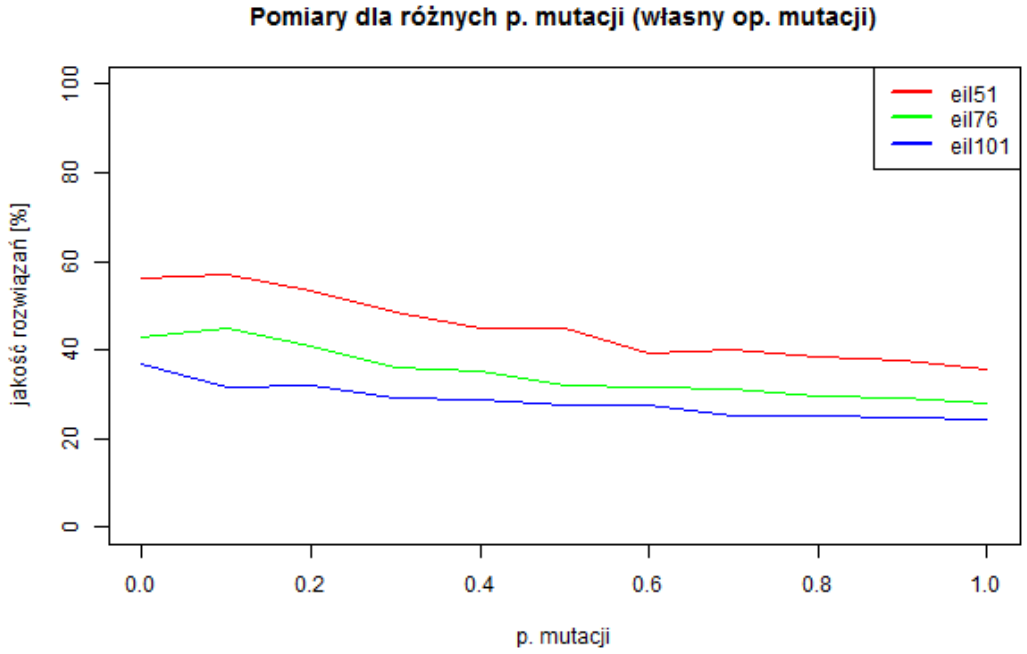
Na ilustracji (rys. 7) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji.



Rysunek 7: Jakość rozwiązań dla różnych wartości p. mutacji

Z powyższego wykresu nie można ustalić wpływu prawdopodobieństwa mutacji na jakość rozwiązań dla instancji "eli76" i "eli101". Dla instancji "eli51" jakość niestabilnie spada wraz ze wzrostem prawdopodobieństwa mutacji.

Na ilustracji (rys. 8) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji z niestandardowym operatorem.



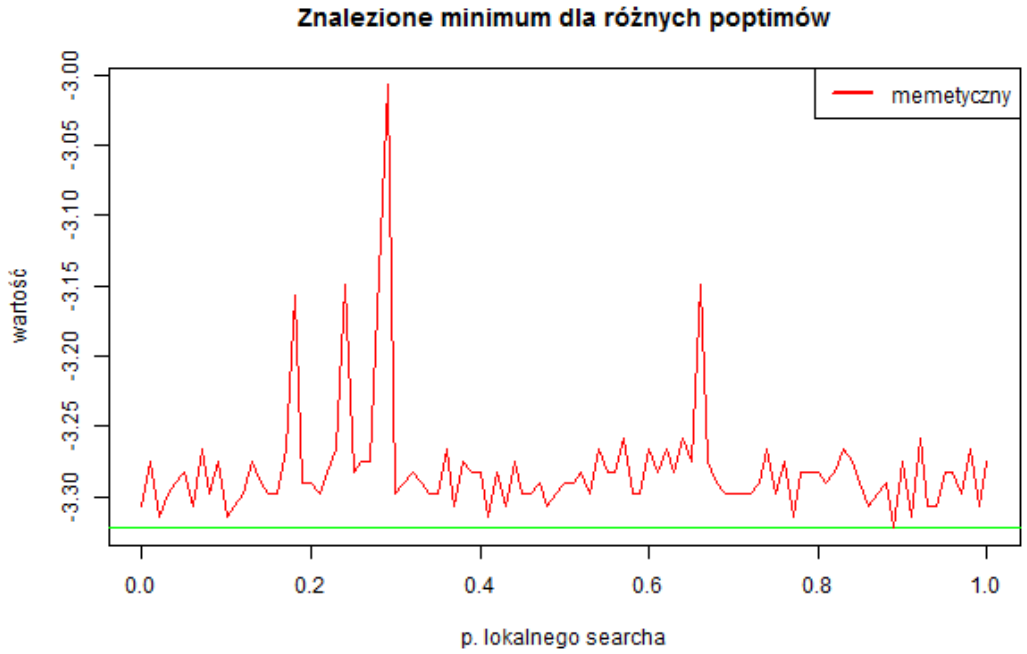
Rysunek 8: Jakość rozwiązań dla różnych wartości p. mutacji (dla własnego operatora)

W przypadku własnej funkcji mutacji jakość rozwiązań problemu komiwojażera widocznie pogarsza u wszystkich instancji wraz ze wzrostem prawdopodobieństwa mutacji. W przypadku "eli51" jest to nawet 20 punktów procentowych.

5 Badania algorytmów dla różnych wartości ich unikalnych parametrów

Algorytmy memetyczny i PSO posiadają własne wartości unikalne, których zmiana może wpłynąć na otrzymywane wyniki. Poniżej przedstawiono wykresy przedstawiające otrzymane wartości dla zmieniających się wybranych parametrów.

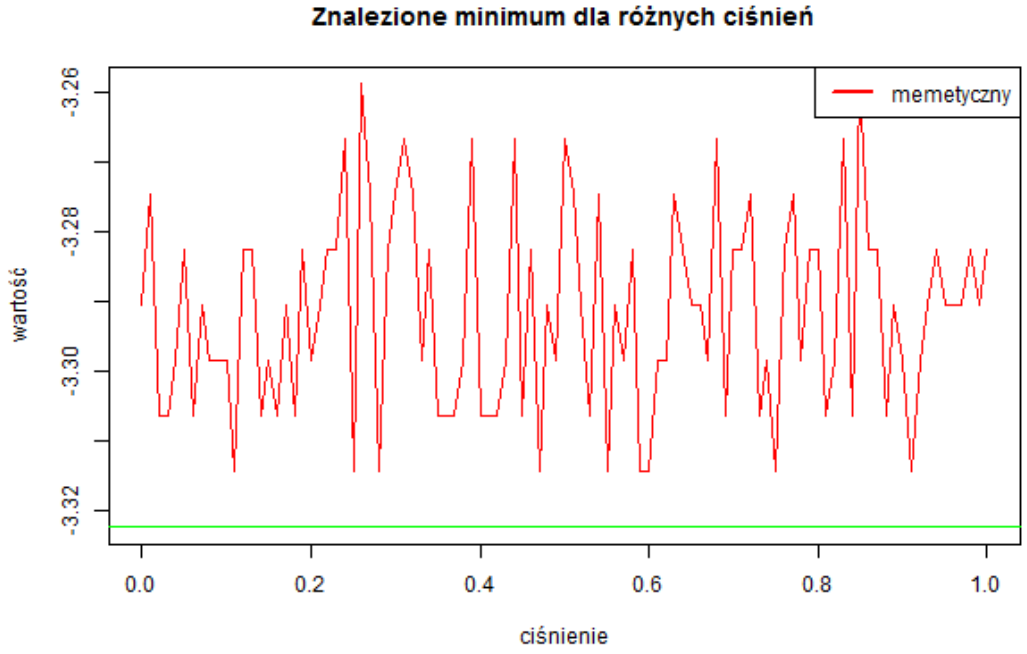
Badania przeprowadzono w przedziale 0-1 z krokiem co 0,01. Otrzymane wyniki są uśrednionymi z 15 iteracji.



Rysunek 9: Jakość rozwiązań dla różnych wartości poptimum algorytmu hybrydowego

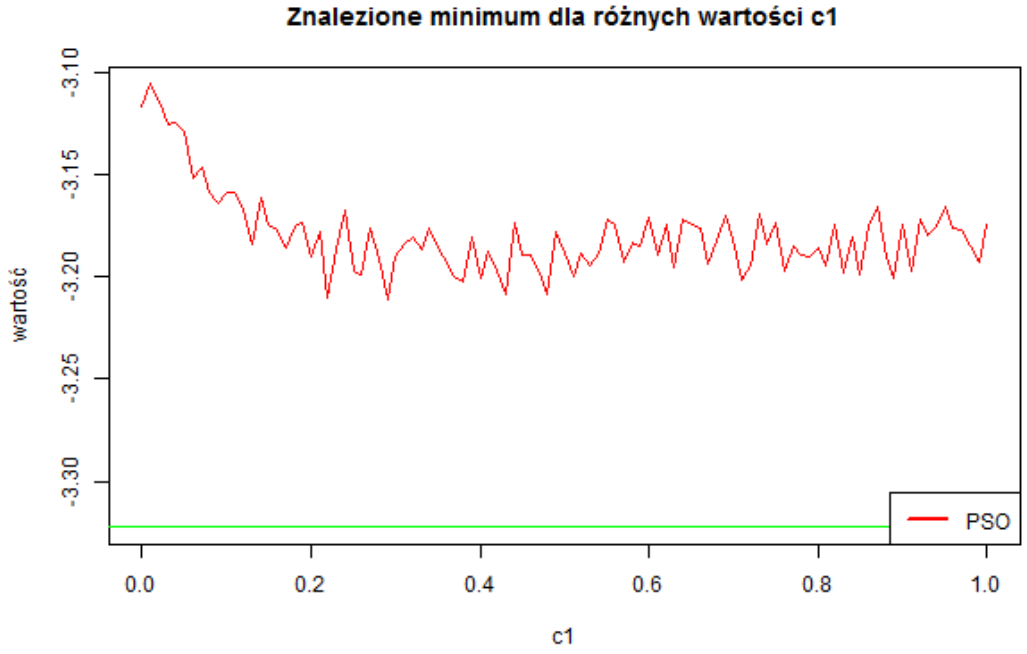
Z wykresu (rys. 9) można odczytać niski wpływ wartości poptimum na otrzymane wyniki. Otrzymywane wartości różnią się nie więcej niż o 0,30.

Zauważalne są 4 skoki zawyżające skalę rezultatów spowodowane heurystyką algorytmu.



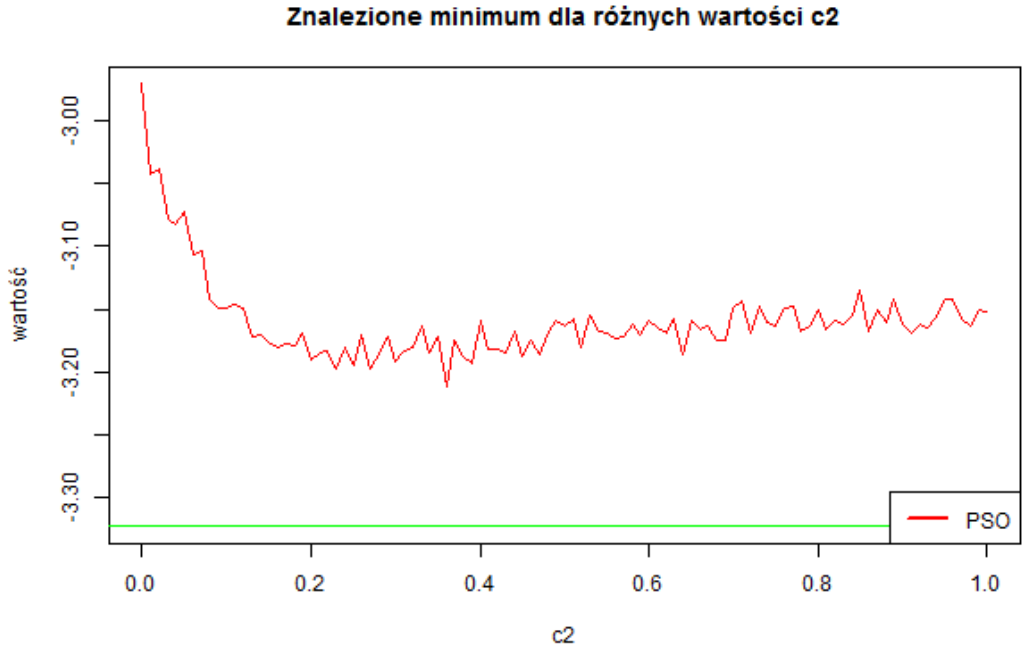
Rysunek 10: Jakość rozwiązań dla różnych wartości ciśnienia algorytmu hybrydowego

Wartości na wykresie (rys. 10) różnią się od siebie o nie więcej niż 0,06. Oznacza to niski wpływ ciśnienia na rezultat algorytmu. Nie zauważalne są trendy wartości wraz ze wzrostem ciśnienia.



Rysunek 11: Jakość rozwiązań dla różnych wartości c1 algorytmu PSO

Z przeprowadzonych badań najlepsze rezultaty otrzymano dla wartości c1 przekraczających 0,2. Po jej różnice spowodowane są heurystyką algorytmu, a nie parametrem.



Rysunek 12: Jakość rozwiązań dla różnych wartości c2 algorytmu PSO

Tak jak w przypadku parametru c1 najlepsze rezultaty uzyskano po przekroczeniu wartości 0,2. Jednak w tym wypadku można zauważyć powolne pogorszenia się wyników im większy parametr c2.

6 Podsumowanie

W trakcie prowadzonych badań przetestowano algorytmy w wariantach genetyczny prosty, genetyczny prosty z własną funkcją mutacji, hybrydowy prosty wraz z własną funkcją, TSP oraz PSO.

Zmiana funkcji mutacji nie spowodowała znaczących zmian w jakości otrzymywanych rozwiązań. Jej działanie jest porównywalne z zaimplementowaną funkcją.

Własna funkcja mutacji dla problemu TSP pogorszyła rezultaty o ok. 10 punktów procentowych.

Z przeprowadzonych badań wynika brak lub niski wpływ zmodyfikowanej funkcji mutacji na działanie badanych algorytmów.

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)