

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

<b>1</b>	<b>Wprowadzenie</b>	<b>2</b>
<b>2</b>	<b>Implementacja</b>	<b>2</b>
2.1	Opis własnych operatorów . . . . .	12
<b>3</b>	<b>Przebieg badań dla problemu optymalizacji rzeczywistej Hartman6</b>	<b>12</b>
<b>4</b>	<b>Przebieg badań dla problemu komiwojażera</b>	<b>16</b>
<b>5</b>	<b>Badania algorytmów dla różnych wartości ich unikalnych parametrów</b>	<b>18</b>
<b>6</b>	<b>Podsumowanie</b>	<b>20</b>

# 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 własnym 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

```
1 # initialize ----
2 # clean old data
3 rm(list=ls())
4 dev.off(dev.list()[ "RStudioGD" ])
5
6 # load libraries
7 require("GA")
8 require("globalOptTests")
9 require("rgl")
10 require("psoptim")
11
12 # custom functions ----
13 # mutation function
14 myMutationFunction <- function(object, parent) {
15   # get GA population
16   population <- parent <- as.vector(object@population[parent, ])
17
18   # calculate randoms
19   rnd <- sample(1:length(population), 1)
20
21   # get min and max from population vector
22   min_value <- which.min(population)
23
24   # set random element to min value
25   population[rnd] = min_value
26 }
```

```

27 |   return (population);
28 | }
29 |
30 | # Settings ----
31 |
32 | nOfRuns <- 1 # 30 number of runs to calc avg scores
33 |
34 | # colors and titles for plot series
35 | colors <- c("red", "purple")
36 | series <- c("GA", "GA + własna mutacja")
37 |
38 | GAWithHybridSeries <- c("GA", "GA + własna mutacja", "Mem", "Mem + własna
    mutacja")
39 | GAWithHybridColors <- c("red", "purple", "blue", "orange")
40 |
41 | # name of function from globalOptTests package
42 | funcName <- "Hartman6"
43 |
44 | # graph settings
45 | graphs <- TRUE #true if you want to print graphs
46 | quality <- 100 #number of probes
47 |
48 | #hybrid algorithm settings
49 | poptim = 0.05 #a value [0,1] specifying the probability of performing a local
    search at each iteration of GA (def 0.1)
50 | pressel = 0.5 #a value [0,1] specifying the pressure selection (def 0.5)
51 |
52 | # Processing ----
53 |
54 | {
55 |   # get data from globalOptTests package
56 |   dim <- getProblemDimen(funcName)
57 |   B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
58 |   f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
    fnName=funcName, checkDim = TRUE)
59 |   globalOpt <- getGlobalOpt(funcName)
60 |
61 |
62 |   if (graphs) {
63 |     # prepare overview graph
64 |     xprobes <- abs(B[2,1] - B[1,1]) / quality
65 |     yprobes <- abs(B[2,2] - B[1,2]) / quality
66 |     x <- seq(B[1,1], B[2,1], by = xprobes)
67 |     y <- seq(B[1,2], B[2,2], by = yprobes)
68 |     z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
69 |     png(file = paste(funcName, "_overview.png", sep=""), width=600, height=400,
    units="px")
70 |     persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
71 |     dev.off()
72 |   }
73 | }
74 |
75 | #GA
76 | customGAMeasure <- function(values, mType, xlab, main) {
77 |
78 |   # main measurement loop (for each serie and sequence calculate average
    results)

```

```

79 temp <- c()
80 for (serie in 1:length(series)) {
81   averages <- c()
82   for (value in values) {
83     sum <- 0
84     for (i in 1:nOfRuns) {
85
86       message(paste("Seria: ", serie))
87       message(paste("Sekwencja: ", value))
88       message(paste("Przebieg: ", i))
89
90       GAmin <- ga(type = "real-valued",
91         mutation = if (serie == 2) myMutationFunction else
92           gaControl("real-valued")$mutation,
93         fitness = function(xx) -f(xx),
94         min = c(B[1,]), max = c(B[2,]),
95         popSize = if (mType == "pop") value else 50,
96         pmutation = if (mType == "mut") value else 0.1)
97       solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
98       eval <- f(solution[1,])
99       sum <- sum + eval
100     }
101     averages <- c(averages, (sum / nOfRuns))
102   }
103   temp <- c(temp, averages)
104 }
105 result <- matrix(c(temp), nrow = length(series), ncol = length(values))
106
107 if (graphs) {
108   # save graph with measurement series to file
109   png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
110     units="px")
111   plot(0, 0, main=main,
112     ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
113     xlim=c(min(values),max(values)),
114     type="n", xlab=xlab, ylab="wartosc")
115   abline(globalOpt,0, col="green")
116   colorNames <- c()
117   seriesNames <- c()
118   for (i in 1:length(series)) {
119     color <- colors[i]
120     colorNames <- c(colorNames, color)
121     seriesNames <- c(seriesNames, series[i])
122     lines(values, result[i,], col = color, type = 'l')
123   }
124   legend("topright", seriesNames, lwd=rep(2,length(series)),
125     lty=rep(1,length(series)), col=colorNames)
126   dev.off()
127 }
128
129 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                 poplim = 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,
157       units="px")
158   plot(0, 0, main=main,
159        ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
160        xlim=c(min(values),max(values)),
161        type="n", xlab=xlab, ylab="wartość")
162   abline(globalOpt,0, col="green")
163   lines(values, averages, col = "red", type = 'l')
164   legend("topright", c("memetyczny"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
165   dev.off()
166 }
167
168 customMeasureGAWithHybrid <- function(values, mType, xlab, main) {
169
170   # main measurement loop (for each serie and sequence calculate average
171   # results)
172   temp <- c()
173   for (serie in 1:length(GAWithHybridSeries)) {
174     averages <- c()
175     for (value in values) {
176       sum <- 0
177       for (i in 1:nOfRuns) {
178
179         message(paste("Seria: ", GAWithHybridSeries[serie]))
180         message(paste("Sekwencja: ", value))
181         message(paste("Przebieg: ", i))
182
183         if(GAWithHybridSeries[serie] == "GA" || GAWithHybridSeries[serie] == "GA
184           + własna funkcja")
185         {
186           GAmin <- ga(type = "real-valued",

```

```

185         mutation = if (serie == 2) myMutationFunction else
186             gaControl("real-valued")$mutation,
187         fitness = function(xx) -f(xx),
188         min = c(B[1,]), max = c(B[2,]),
189         popSize = if (mType == "pop") value else 50,
190         pmutation = if (mType == "mut") value else 0.1)
191     }
192     else
193     {
194         GAmin <- ga(type = "real-valued",
195             mutation = if (serie == 4) myMutationFunction else
196                 gaControl("real-valued")$mutation,
197             fitness = function(xx) -f(xx),
198             min = c(B[1,]), max = c(B[2,]),
199             optim = TRUE,
200             optimArgs = list (
201                 poplim = if (mType == "poplim") value else 0.05,
202                 pressel = if (mType == "pressel") value else 0.5))
203     }
204
205     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
206     eval <- f(solution[1,])
207     sum <- sum + eval
208 }
209 averages <- c(averages, (sum / nOfRuns))
210 }
211 result <- matrix(c(temp), nrow = length(GAWithHybridSeries), ncol =
212     length(values))
213
214 if (graphs) {
215     # create standalone graph for each serie
216     for (serie in 1:length(GAWithHybridSeries)) {
217         legendColors <- rep("darkslateblue", length(GAWithHybridSeries))
218         legendColors[serie] = "red"
219         # save graph with measurement series to file
220         png(file = paste(funcName, mType, serie, ".png", sep=""), width=600,
221             height=400, units="px")
222         plot(0, 0, main=main,
223             ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
224             xlim=c(min(values),max(values)),
225             type="n", xlab=xlab, ylab="wartosc")
226         abline(globalOpt,0, col="green")
227
228         lastLine <- NA
229         seriesNames <- c()
230         for (i in 1:length(GAWithHybridSeries)) {
231             seriesNames <- c(seriesNames, GAWithHybridSeries[i])
232             if (i != serie)
233             {
234                 lines(values, result[i,], col = "darkslateblue", type = 'l', lwd = 2)
235             }
236         }
237         lines(values, result[serie,], col = "red", type = 'l', lwd = 2)
238     }
239 }

```

```

237     legend("topright", seriesNames, lwd=rep(2,length(GAWithHybridSeries)),
238           lty=rep(1,length(GAWithHybridSeries)), col = legendColors)
239     dev.off()
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   #xmin <- c(-5.12, -5.12)
254   #xmax <- c(5.12, 5.12)
255
256   vmax <- c(rep(4, length(xmin)))
257
258   g <- function(x) {
259
260     vec <- c()
261     for (row in 1:length(x[,1])) {
262       val <- -f(x[row,])
263       vec <- c(vec, val)
264     }
265
266     vec
267
268   }
269
270
271   #g <- function(x) {
272   # -(200 + x[,1]^2 + x[,2]^2 + cos(2*pi*x[,2]))
273   #}
274
275   averages <- c()
276   for (value in values)
277   {
278     sum <- 0
279     for (i in 1:nOfRuns)
280     {
281       message(paste("Sekwencja: ", value))
282       message(paste("Przebieg: ", i))
283
284       result <- psoptim(FUN=g,
285                        n=n,
286                        max.loop=m.l,
287                        w=w,
288                        c1=if (valueType == "c1") value else 0.2,
289                        c2=if (valueType == "c2") value else 0.2,
290                        xmin=xmin,
291                        xmax=xmax,

```



```

292         vmax=vmax,
293         seed=NULL,
294         anim=FALSE)
295
296     sum <- sum + result$val
297
298 }
299 averages <- c(averages, (sum / nOfRuns))
300 }
301
302 if (graphs) {
303
304     # save graph with measurement series to file
305     png(file = paste(funcName, valueType, ".png", sep=""), width=600,
306         height=400, units="px")
307     plot(0, 0, main=title,
308         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
309         xlim=c(min(values),max(values)),
310         type="n", xlab=xLabel, ylab="wartość")
311     abline(globalOpt,0, col="green")
312     lines(values, averages, col = "red", type = 'l')
313     legend("bottomright", c("PSO"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
314     dev.off()
315 }
316
317
318
319 # perform set of measurements ----
320 # GA
321 customGAMeasure(seq(0, 1, 0.1), "mut",
322                 "p. mutacji", "Znalezione minimum dla różnych p. mutacji")
323
324 customGAMeasure(seq(10, 100, 10), "pop",
325                 "rozmiar populacji", "Znalezione minimum dla różnych rozmiarów
326                 populacji")
327
328 # Hybrid
329 customHybridMeasure(seq(0, 1, 0.01), "poptim",
330                        "p. lokalnego searcha", "Znalezione minimum dla różnych
331                        poptimów")
332
333 customHybridMeasure(seq(0, 1, 0.01), "pressel",
334                        "ciśnienie", "Znalezione minimum dla różnych ciśnień")
335
336 # Mixed (GA+Hybrid)
337 customMeasureGAWithHybrid(seq(0, 1, 0.01), "mut",
338                            "p.mutacji", "Znalezione minimum dla różnych p. mutacji")
339
340 # PSO
341 customPSOMeasure(seq(0, 1, 0.1), "c1",
342                  "c1", "Znalezione minimum dla różnych wartości c1")

```

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.

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 częściom. 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

```
1 # clean old data
2 rm(list=ls())
3 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 <- 1 #15
13
14 # instances to test and best known solutions
15 instances <- c("eil51", "eil76", "eil101")
16 best_solutions <- c(426, 538, 629)
17 colors <- c("red", "green", "blue")
18
19 tourLength <- function(tour, distMatrix) {
20   tour <- c(tour, tour[1])
21   route <- embed(tour, 2)[,2:1]
22   sum(distMatrix[route])
23 }
```

```

23 }
24
25 fit <- function(tour, distMatrix) 1/tourLength(tour, distMatrix)
26
27 customMutation <- function(object, parent, ...) {
28
29   # Insertion mutation
30   parent <- as.vector(object@population[parent,])
31   n <- length(parent)
32   m <- sample(1:n, size = 1)
33   pos <- sample(1:(n-1), size = 1)
34   i <- c(setdiff(1:pos,m), m, setdiff((pos+1):n,m))
35   mutate <- parent[i]
36
37   # Displacement mutation
38   parent <- mutate
39   m <- sort(sample(1:n, size = 2))
40   m <- seq(m[1], m[2], by = 1)
41   l <- max(m)-min(m)+1
42   pos <- sample(1:max(1,(n-1)), size = 1)
43   i <- c(setdiff(1:n,m)[1:pos], m, setdiff(1:n,m)[- (1:pos)])
44   mutate <- parent[na.omit(i)]
45
46   # Scramble mutation
47   parent <- mutate
48   m <- sort(sample(1:n, size = 2))
49   m <- seq(min(m), max(m), by = 1)
50   m <- sample(m, replace = FALSE)
51   i <- c(setdiff(1:min(m),m), m, setdiff(max(m):n,m))
52   mutate <- parent[i]
53   return(mutate)
54 }
55
56
57 performTest <- function(testName, graphMain, graphXLab,
58                         sequenceType, sequence,
59                         popsize=50, pcrossover=0.8,
60                         pmutation=0.1, maxiter=100, mutation = NULL) {
61
62   solution_qualities <- c()
63
64   # each instance as separate serie
65   for (i in 1:length(instances)) {
66
67     fileName = paste("examples/", instances[i], ".tsp", sep="")
68     graphTitle = paste("TSPLIB: ", instances[i], sep="")
69
70     drill <- read_TSPLIB(system.file(fileName, package = "TSP"))
71     D <- as.matrix(dist(drill, method = "euclidean"))
72     N <- max(dim(D))
73
74     solution_quality <- c()
75
76     bestTour <- NA
77     bestTourLength <- .Machine$integer.max
78     averageLength <- 0

```

```

79
80   for (s in 1:length(sequence)) {
81
82     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
94                         popsize,
95               pcrossover = if (sequenceType == "pcrossover") sequence[s] else
96                           pcrossover,
97               pmutation = if (sequenceType == "pmutation") sequence[s] else
98                           pmutation,
99               maxiter = if (sequenceType == "maxiter") sequence[s] else
100                        maxiter,
101               mutation = if (is.null(mutation))
102                          gaControl("permutation")$mutation else mutation)
103
104       tour <- GA@solution[1, ]
105       tl <- tourLength(tour, D)
106
107       if (tl < bestTourLength) {
108         bestTourLength <- tl
109         bestTour <- tour
110       }
111
112       averageLength <- averageLength + (tl - averageLength) / n
113
114     }
115
116     solution_quality <- c(solution_quality,
117                          (best_solutions[i]/averageLength) * 100)
118
119   }
120
121   png(file = paste(testName, "_", instances[i], ".png", sep=""), width=600,
122       height=400, units="px")
123   plot(drill, bestTour, cex=.6, col = "red", pch=3, main = graphTitle)
124   dev.off()
125
126   solution_qualities <- c(solution_qualities, solution_quality)
127
128 }
129
130 qualities = matrix(solution_qualities,
131                   nrow=length(instances), ncol=length(sequence), byrow = TRUE)
132
133 # 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)),
133      type="n", xlab=graphXLab, ylab="jakość rozwiązań [%]")
134 for (i in 1:length(instances)) {
135     lines(sequence, qualities[i,], col = colors[i], type = 'l')
136 }
137 legend("topright", instances, lwd=rep(2,length(instances)),
138       lty=rep(1,length(instances)), col=colors)
139 dev.off()
140 }
141
142 performTest(testName = "tsp_pop",
143             graphMain = "Pomiary dla różnych rozmiarów populacji",
144             graphXLab = "rozmiar populacji",
145             sequenceType = "popsize", sequence = seq(50, 500, 50))
146
147 performTest(testName = "tsp_mut",
148             graphMain = "Pomiary dla różnych p. mutacji",
149             graphXLab = "p. mutacji",
150             sequenceType = "pmutation", sequence = seq(0, 1, 0.1))
151
152 performTest(testName = "tsp_mut_custom",
153             graphMain = "Pomiary dla różnych p. mutacji (własny op. mutacji)",
154             graphXLab = "p. mutacji",
155             sequenceType = "pmutation", sequence = seq(0, 1, 0.1), mutation =
              customMutation)

```

## 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.

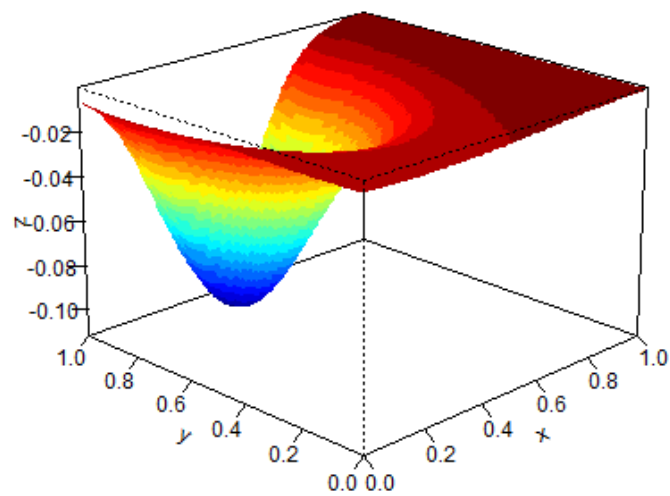
## 3 Przebieg badań dla problemu optymalizacji rzeczywistej Hartman6

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. ??) przedstawiono jej wykres dla pierwszych dwóch. Poniżej zamieszczono jej wzór (1).

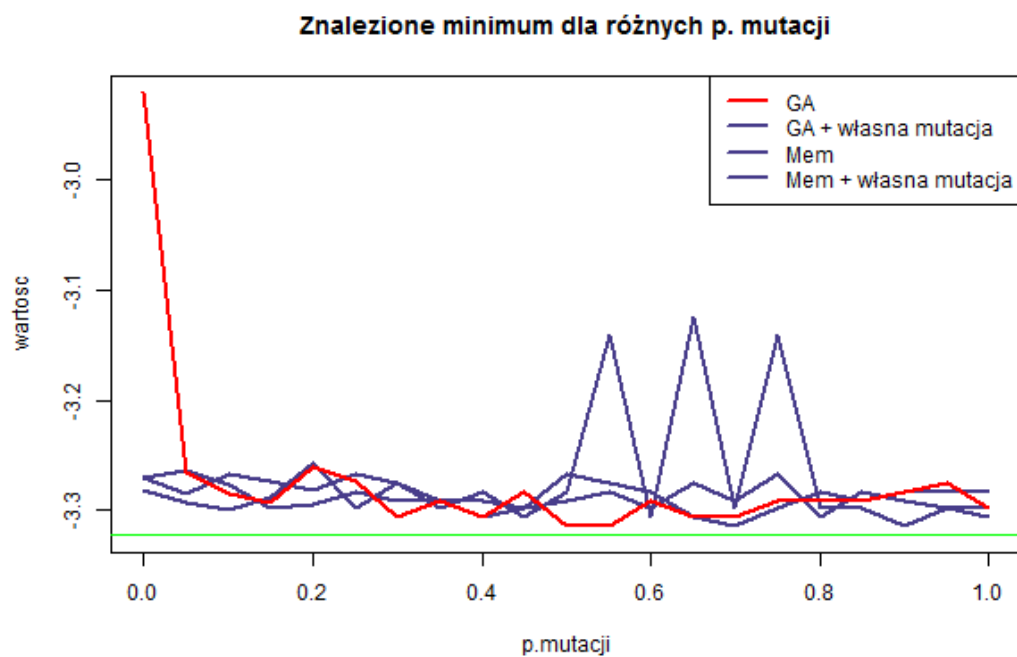
$$f(\mathbf{x}) = - \sum_{i=1}^4 c_i \exp \left[ - \sum_{j=1}^6 a_{ij} (x_j - p_{ij})^2 \right] \quad (1)$$

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

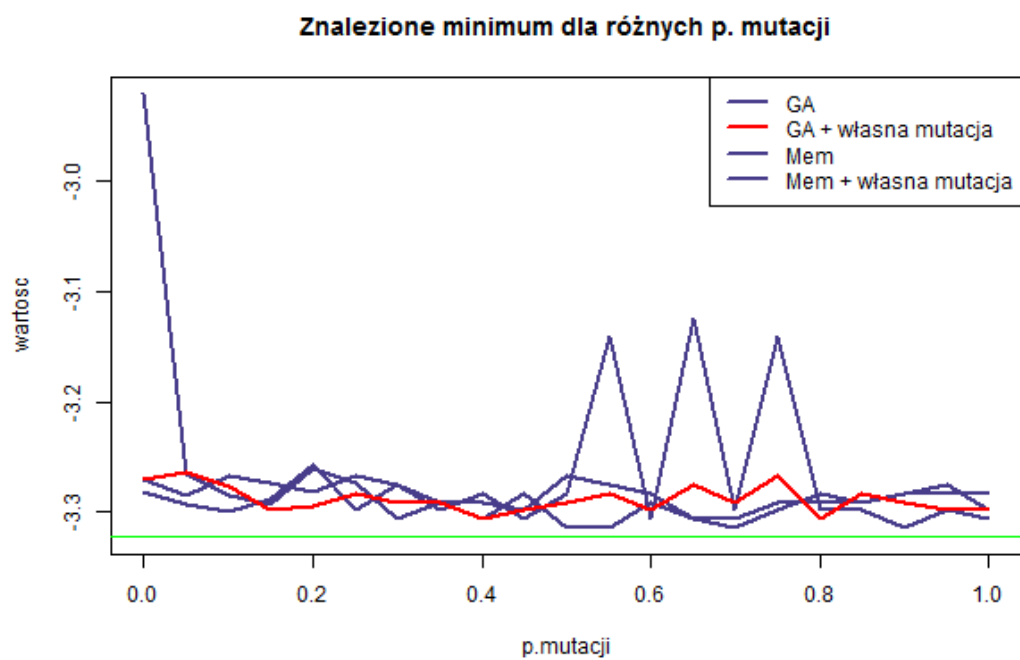


Rysunek 1: Wykres funkcji Hartman6

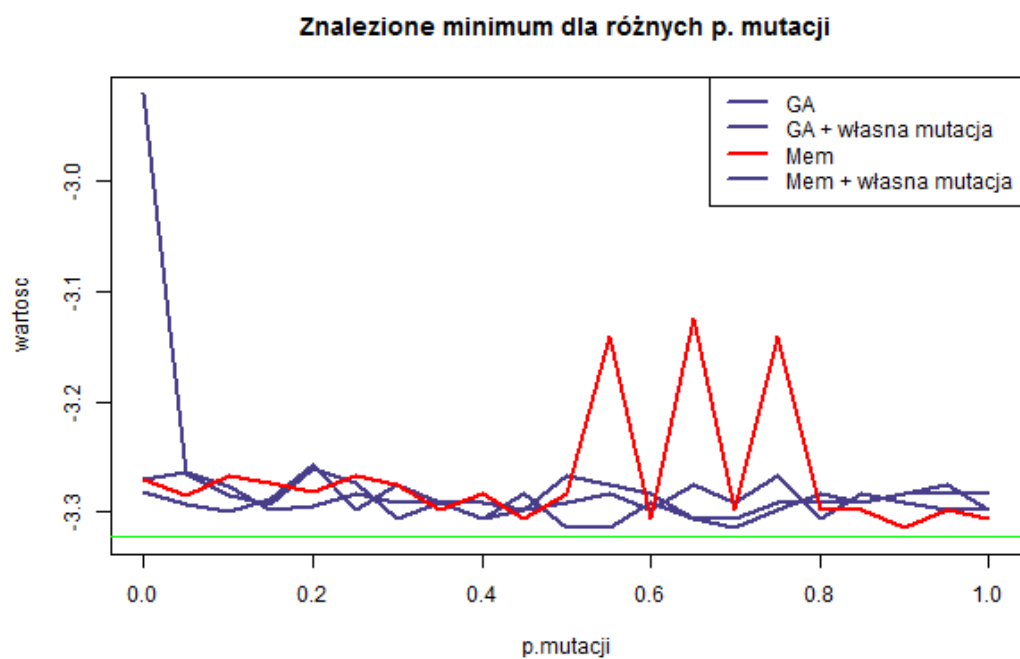
Na kolejnych stronach zamieszczono wyniki badań porównawczych mutacji przeprowadzonych na algorytmie.



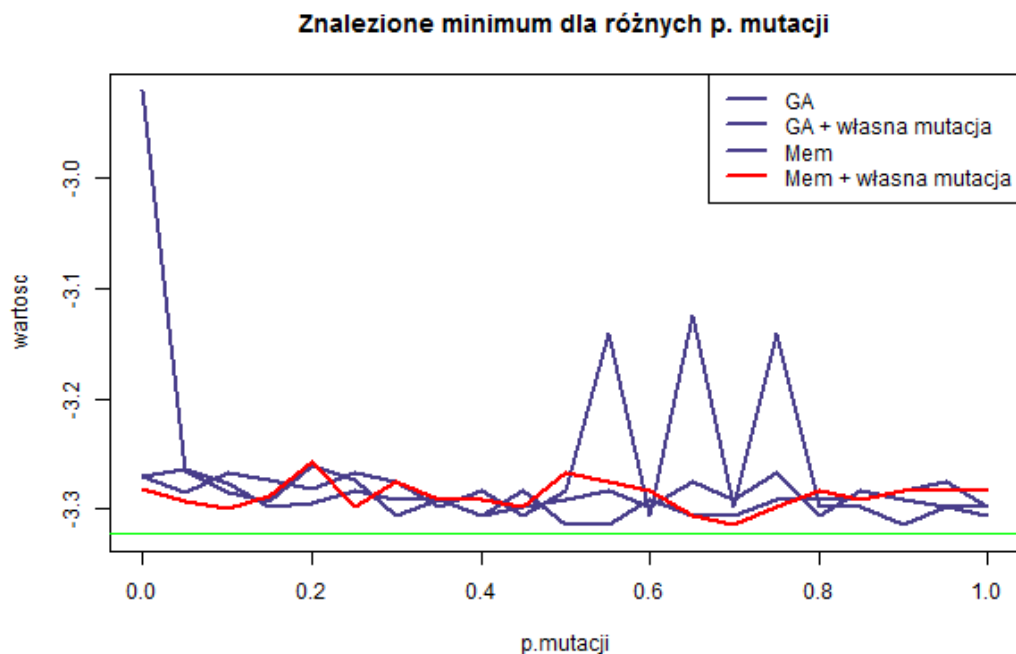
Rysunek 2: Wykres funkcji Hartman6



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

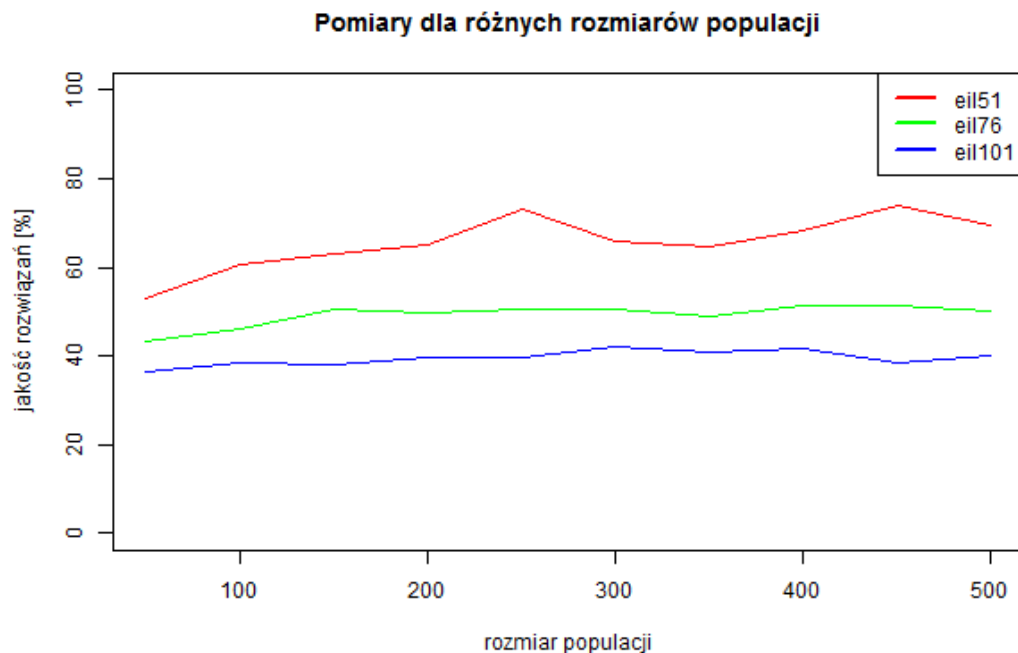
Przeprowadzono badania z zakresu optymalizacji marszruty dla problemu komiwojażera. Wykorzystano trzy instancje problemu z biblioteki TSPLIB:

- eil51
- eil76
- eil101

Jakość rozwiązań wyraża się wzorem:

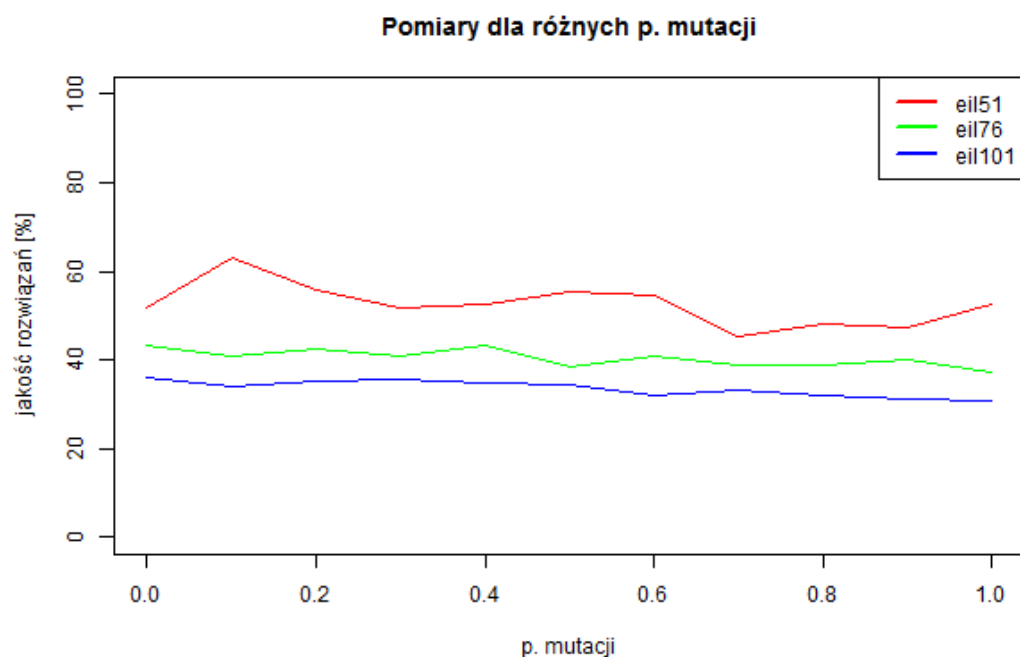
$$quality\ of\ solution = \frac{shortest\ known\ path}{found\ path} * 100\% \quad (2)$$

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



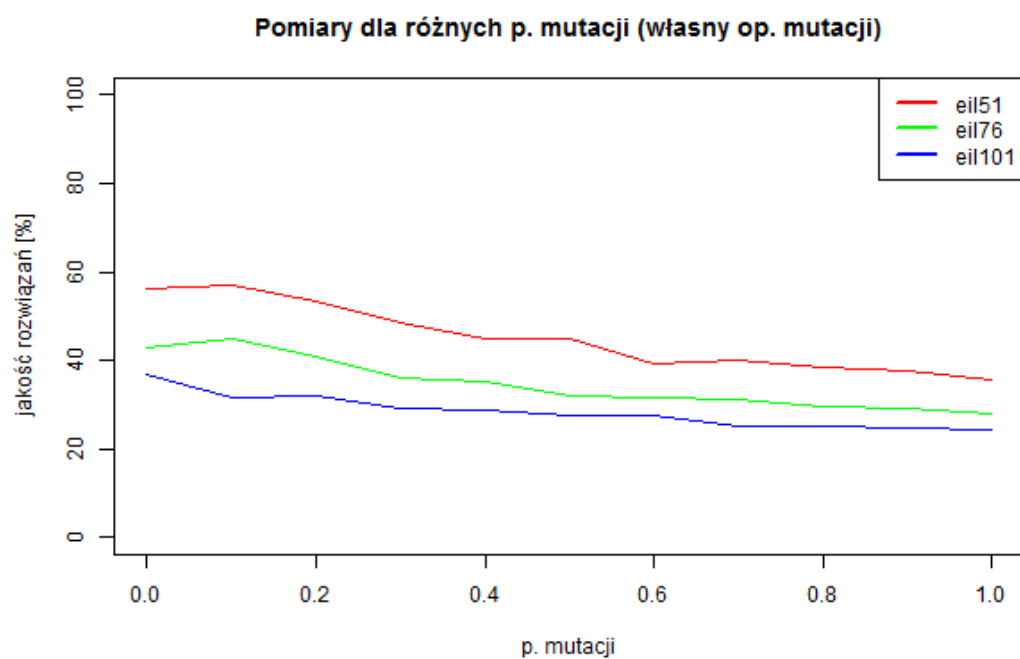
Rysunek 6: Jakość rozwiązań dla różnych rozmiarów populacji

Na ilustracji (rys. 10) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji.



Rysunek 7: Jakość rozwiązań dla różnych wartości p. mutacji

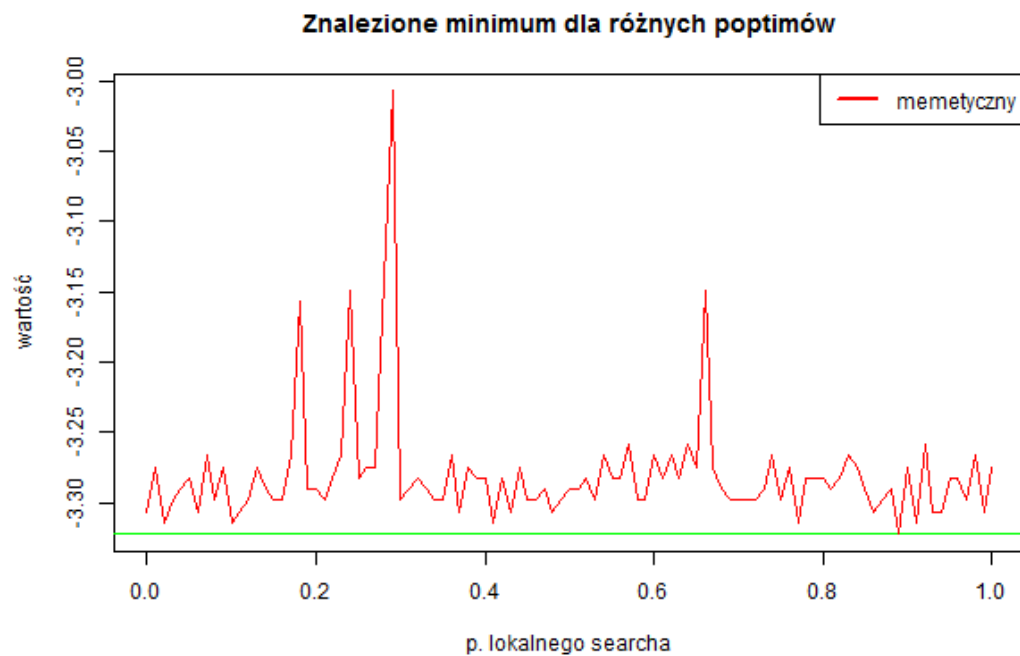
Na ilustracji (rys. 8) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji z niestandardowym operatorem.



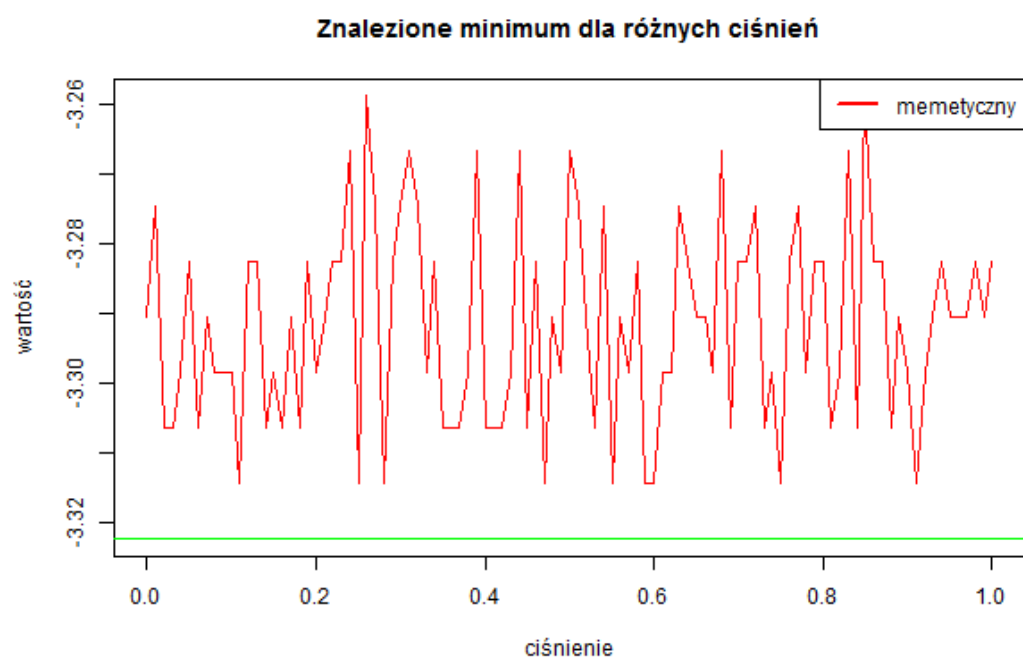
Rysunek 8: Jakość rozwiązań dla różnych wartości p. mutacji (dla własnego operatora)

## 5 Badania algorytmów dla różnych wartości ich unikalnych parametrów

Wstęp[todo]



Rysunek 9: Jakość rozwiązań dla różnych wartości poptimum



Rysunek 10: Jakość rozwiązań dla różnych wartości ciśnienia

## 6 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)