

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	2
2	Implementacja	2
2.1	Opis własnych operatorów	12
3	Przebieg badań dla problemu optymalizacji rzeczywistej Hartman6	12
4	Przebieg badań dla problemu komiwojażera	15
5	Podsumowanie	17

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 <- 15 # 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 | customGAMeasure <- function(values, mType, xlab, main) {
55 |
56 |   # main measurement loop (for each serie and sequence calculate average
    results)
57 |   temp <- c()
58 |   for (serie in 1:length(series)) {
59 |     averages <- c()
60 |     for (value in values) {
61 |       sum <- 0
62 |       for (i in 1:nOfRuns) {
63 |
64 |         message(paste("Seria: ", serie))
65 |         message(paste("Sekwencja: ", value))
66 |         message(paste("Przebieg: ", i))
67 |
68 |         GAmin <- ga(type = "real-valued",
69 |           mutation = if (serie == 2) myMutationFunction else
            gaControl("real-valued")$mutation,
70 |           fitness = function(xx) -f(xx),
71 |           min = c(B[1,]), max = c(B[2,]),
72 |           popSize = if (mType == "pop") value else 50,
73 |           pmutation = if (mType == "mut") value else 0.1)
74 |         solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
75 |         eval <- f(solution[1,])
76 |         sum <- sum + eval
77 |       }
78 |       averages <- c(averages, (sum / nOfRuns))

```

```

79     }
80     temp <- c(temp, averages)
81 }
82 result <- matrix(c(temp), nrow = length(series), ncol = length(values))
83
84 if (graphs) {
85
86     # save graph with measurement series to file
87     png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
88         units="px")
89     plot(0, 0, main=main,
90         ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
91         xlim=c(min(values),max(values)),
92         type="n", xlab=xlab, ylab="wartosc")
93     abline(globalOpt,0, col="green")
94     colorNames <- c()
95     seriesNames <- c()
96     for (i in 1:length(series)) {
97         color <- colors[i]
98         colorNames <- c(colorNames, color)
99         seriesNames <- c(seriesNames, series[i])
100         lines(values, result[i,], col = color, type = 'l')
101     }
102     legend("topright", seriesNames, lwd=rep(2,length(series)),
103         lty=rep(1,length(series)), col=colorNames)
104     dev.off()
105 }
106
107 customMeasureGAWithHybrid <- function(values, mType, xlab, main) {
108
109     # main measurement loop (for each serie and sequence calculate average
110     # results)
111     temp <- c()
112     for (serie in 1:length(GAWithHybridSeries)) {
113         averages <- c()
114         for (value in values) {
115             sum <- 0
116             for (i in 1:nOfRuns) {
117
118                 message(paste("Seria: ", GAWithHybridSeries[serie]))
119                 message(paste("Sekwencja: ", value))
120                 message(paste("Przebieg: ", i))
121
122                 if(GAWithHybridSeries[serie] == "GA" || GAWithHybridSeries[serie] == "GA
123                     + własna funkcja")
124                 {
125                     GAmin <- ga(type = "real-valued",
126                         mutation = if (serie == 2) myMutationFunction else
127                             gaControl("real-valued")$mutation,
128                         fitness = function(xx) -f(xx),
129                         min = c(B[1,]), max = c(B[2,]),
130                         popSize = if (mType == "pop") value else 50,
131                         pmutation = if (mType == "mut") value else 0.1)

```

```

130     else
131     {
132         GAmin <- ga(type = "real-valued",
133             mutation = if (serie == 4) myMutationFunction else
134                 gaControl("real-valued")$mutation,
135             fitness = function(xx) -f(xx),
136             min = c(B[1,]), max = c(B[2,]),
137             optim = TRUE,
138             optimArgs = list (
139                 poptim = if (mType == "poptim") value else 0.05,
140                 pressel = if (mType == "pressel") value else 0.5))
141     }
142
143     solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
144     eval <- f(solution[1,])
145     sum <- sum + eval
146 }
147 averages <- c(averages, (sum / nOfRuns))
148 }
149 temp <- c(temp, averages)
150 }
151 result <- matrix(c(temp), nrow = length(GAWithHybridSeries), ncol =
152     length(values))
153
154 if (graphs) {
155     # create standalone graph for each serie
156     for (serie in 1:length(GAWithHybridSeries)) {
157         legendColors <- rep("darkslateblue", length(GAWithHybridSeries))
158         legendColors[serie] = "red"
159         # save graph with measurement series to file
160         png(file = paste(funcName, mType, serie, ".png", sep=""), width=600,
161             height=400, units="px")
162         plot(0, 0, main=main,
163             ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
164             xlim=c(min(values),max(values)),
165             type="n", xlab=xlab, ylab="wartosc")
166         abline(globalOpt,0, col="green")
167
168         lastLine <- NA
169         seriesNames <- c()
170         for (i in 1:length(GAWithHybridSeries)) {
171             seriesNames <- c(seriesNames, GAWithHybridSeries[i])
172             if (i != serie)
173             {
174                 lines(values, result[i,], col = "darkslateblue", type = 'l', lwd = 2)
175             }
176         }
177         lines(values, result[serie,], col = "red", type = 'l', lwd = 2)
178
179         legend("topright", seriesNames, lwd=rep(2,length(GAWithHybridSeries)),
180             lty=rep(1,length(GAWithHybridSeries)), col = legendColors)
181         dev.off()
182     }
183 }
184 }
185 }

```

```

182 {
183 # get data from globalOptTests package
184 dim <- getProblemDimen(funcName)
185 B <- matrix(unlist(getDefaultBounds(funcName)),ncol=dim,byrow=TRUE)
186 f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
187                           fnName=funcName, checkDim = TRUE)
188 globalOpt <- getGlobalOpt(funcName)
189
190 if (graphs) {
191   # prepare overview graph
192   xprobes <- abs(B[2,1] - B[1,1]) / quality
193   yprobes <- abs(B[2,2] - B[1,2]) / quality
194   x <- seq(B[1,1], B[2,1], by = xprobes)
195   y <- seq(B[1,2], B[2,2], by = yprobes)
196   z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
197   png(file = paste(funcName, "_overview.png", sep=""), width=600, height=400,
198        units="px")
199   persp3D(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
200   dev.off()
201 }
202
203
204 # perform set of measurements ----
205 customMeasureGAWithHybrid(seq(0, 1, 0.01), "mut", "p.mutacji", "Znalezienie
    minimum dla różnych p. mutacji")
206
207 customGAMeasure(seq(0, 1, 0.1), "mut",
208                 "p. mutacji", "Znalezienie minimum dla różnych p. mutacji")
209 customGAMeasure(seq(10, 100, 10), "pop",
210                 "rozmiar populacji", "Znalezienie minimum dla różnych rozmiarów populacji")
211
212
213
214 # hybrid algorithm ----
215
216 customHybridMeasure <- function(values, mType, xlab, main) {
217
218   averages <- c()
219   for (value in values) {
220     sum <- 0
221     for (i in 1:nOfRuns) {
222
223       message(paste("Sekwencja: ", value))
224       message(paste("Przebieg: ", i))
225
226       GAmin <- ga(type = "real-valued",
227                  fitness = function(xx) -f(xx),
228                  min = c(B[1,]), max = c(B[2,]),
229                  optim = TRUE,
230                  optimArgs = list (
231                    poptim = if (mType == "poptim") value else 0.05,
232                    pressel = if (mType == "pressel") value else 0.5))
233       solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
234       eval <- f(solution[1,])
235       sum <- sum + eval

```

```

236     }
237     averages <- c(averages, (sum / nOfRuns))
238 }
239
240 if (graphs) {
241
242     # save graph with measurement series to file
243     png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
244         units="px")
245     plot(0, 0, main=main,
246         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
247         xlim=c(min(values),max(values)),
248         type="n", xlab=xlab, ylab="wartość")
249     abline(globalOpt,0, col="green")
250     lines(values, averages, col = "red", type = 'l')
251     legend("topright", c("memetyczny"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
252     dev.off()
253 }
254
255 customHybridMeasure(seq(0, 1, 0.05), "poptim",
256     "p. lokalnego searcha", "Znalezione minimum dla różnych poptimów")
257 customHybridMeasure(seq(0, 1, 0.1), "pressel",
258     "ciśnienie", "Znalezione minimum dla różnych ciśnień")
259
260
261 # PSO tests ----
262
263
264 nOfRuns = 1 # zostaje bo niby nie można uśredniać?
265
266
267 #TODO
268 customPSOMeasure <- function(values, mType, xlab, main) {
269
270     averages <- c()
271     for (value in values) {
272         sum <- 0
273         for (i in 1:nOfRuns) {
274
275             message(paste("Sekwencja: ", value))
276             message(paste("Przebieg: ", i))
277
278             GAmin <- ga(type = "real-valued",
279                 fitness = function(xx) -f(xx),
280                 min = c(B[1,]), max = c(B[2,]),
281                 optim = TRUE,
282                 optimArgs = list (
283                     poptim = if (mType == "poptim") value else 0.05,
284                     pressel = if (mType == "pressel") value else 0.5))
285
286             solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
287             eval <- f(solution[1,])
288             sum <- sum + eval
289         }
290     averages <- c(averages, (sum / nOfRuns))

```



```

291 }
292
293 if (graphs) {
294
295     # save graph with measurement series to file
296     png(file = paste(funcName, mType, ".png", sep=""), width=600, height=400,
297         units="px")
298     plot(0, 0, main=main,
299         ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
300         xlim=c(min(values),max(values)),
301         type="n", xlab=xlable, ylab="wartość")
302     abline(globalOpt,0, col="green")
303     lines(values, averages, col = "red", type = 'l')
304     legend("topright", c("memetyczny"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
305     dev.off()
306 }
307
308
309 n <- 500 #ilosc czastek
310 m.l <- 50 #ilosc przebiegow
311 w <- 0.95
312 c1 <- 0.2
313 c2 <- 0.2
314 xmin <- c(-5.12, -5.12)
315 xmax <- c(5.12, 5.12)
316 vmax <- c(4, 4)
317
318 #inaczej są parametry podawane, trzeba zrobić dodatkowego wrappera na f()
319 g <- function(x) {
320     -(200 + x[,1]^2 + x[,2]^2 + cos(2*pi*x[,2]))
321 }
322
323 psoptim(FUN=g, n=n, max.loop=m.l, w=w, c1=c1, c2=c2,
324         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 kolejne 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

```

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 }
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, ]

```

```

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,
117      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)),
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)
138 dev.off()
139
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
155

```
graphXLab = "p. mutacji",  
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 znalezioną minimalną. Gwarantuje to niepojawienie się z populacji wartości nieoczekiwanej, lecz tylko te otrzymane podczas działania algorytmu.

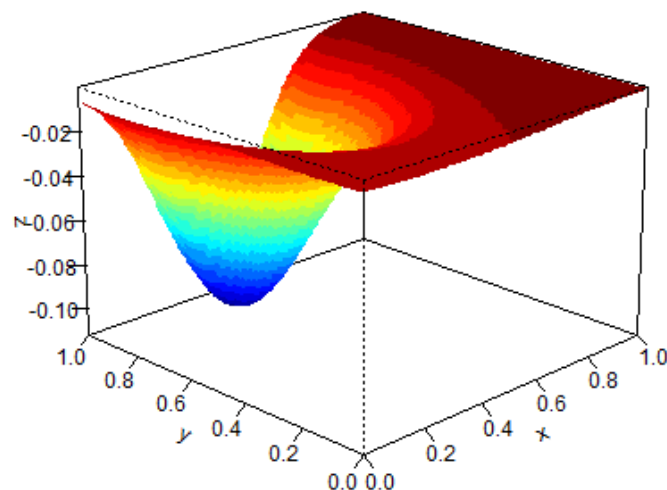
3 Przebieg badań dla problemu optymalizacji rzeczywistej Hartman6

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).

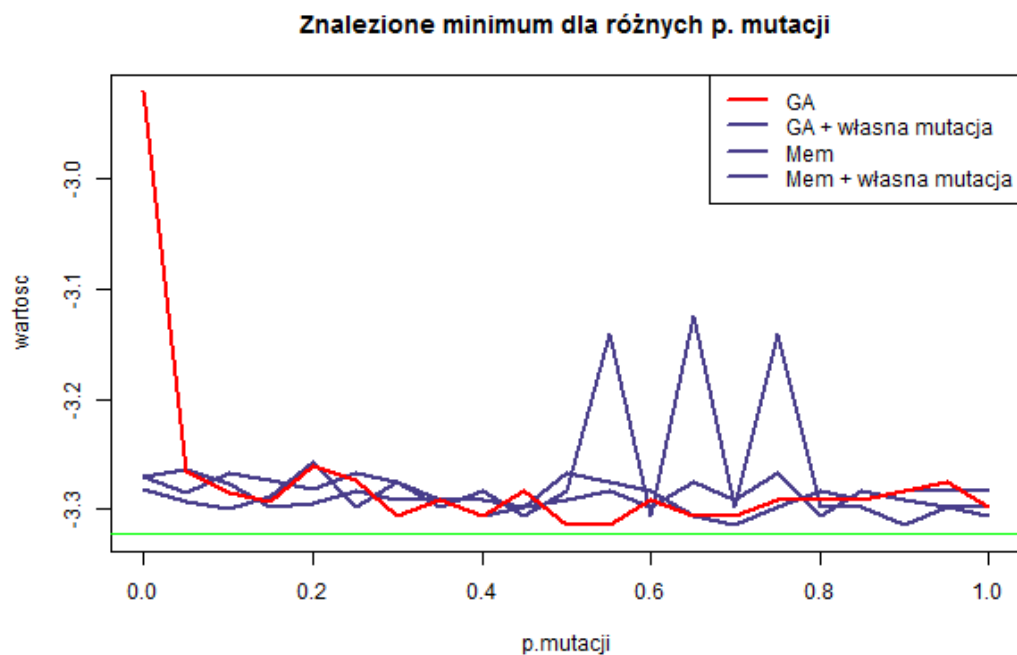
$$f(\mathbf{x}) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^6 a_{ij}(x_j - p_{ij})^2\right] \quad (1)$$

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

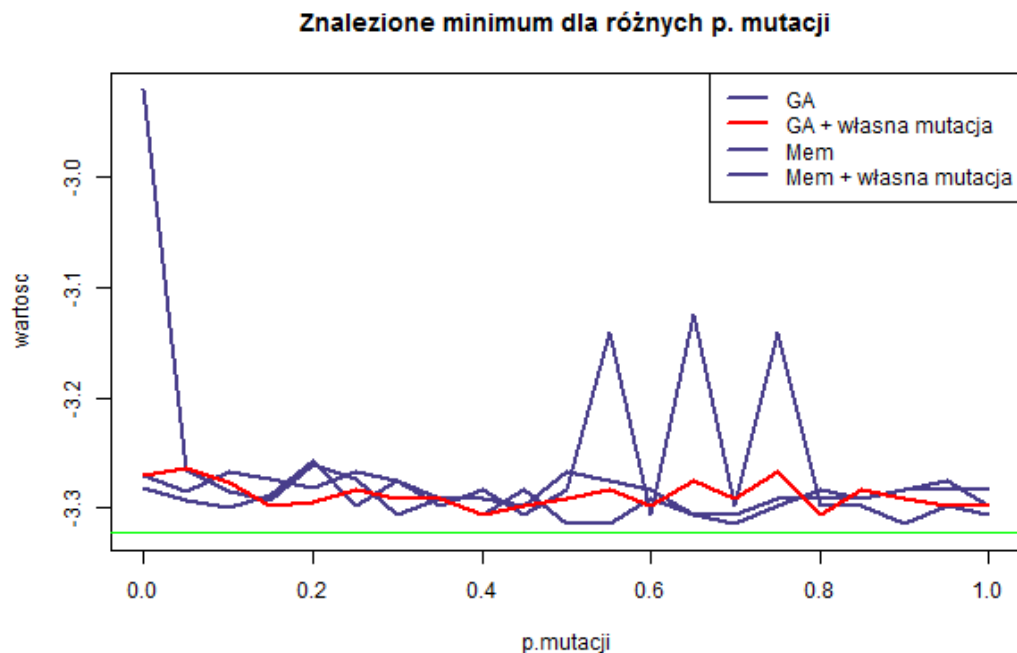


Rysunek 1: Wykres funkcji Hartman6

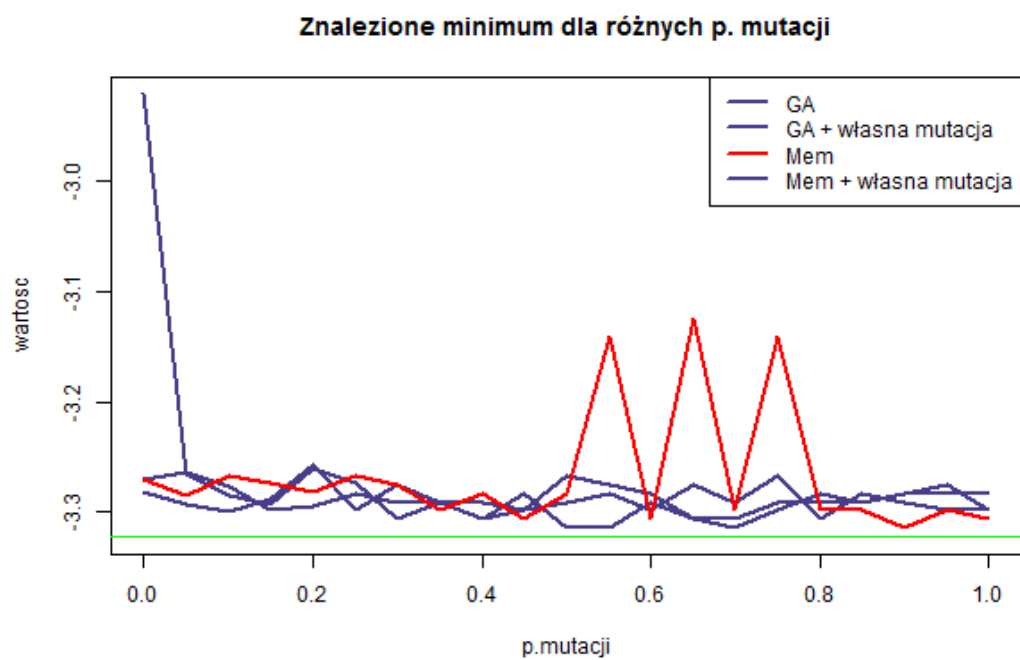
Na kolejnych stronach zamieszczono wyniki badań porównawczych mutacji przeprowadzonych na algorytmie.



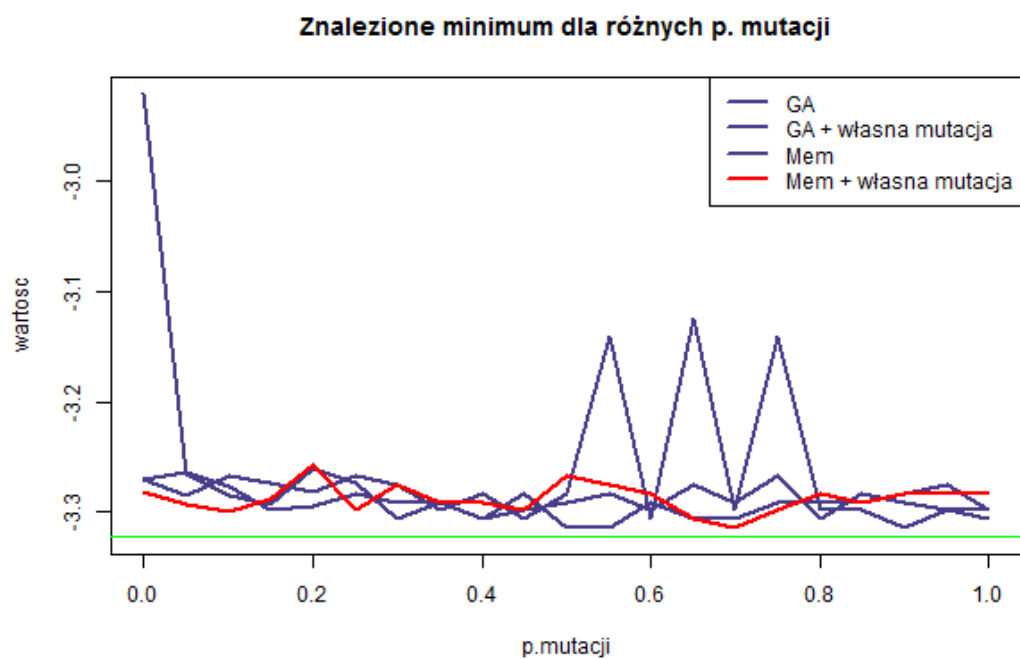
Rysunek 2: Wykres funkcji Hartman6



Rysunek 3: Wykres funkcji Hartman6



Rysunek 4: Wykres funkcji Hartman6



Rysunek 5: Wykres funkcji Hartman6

4 Przebieg badań dla problemu komiwojażera

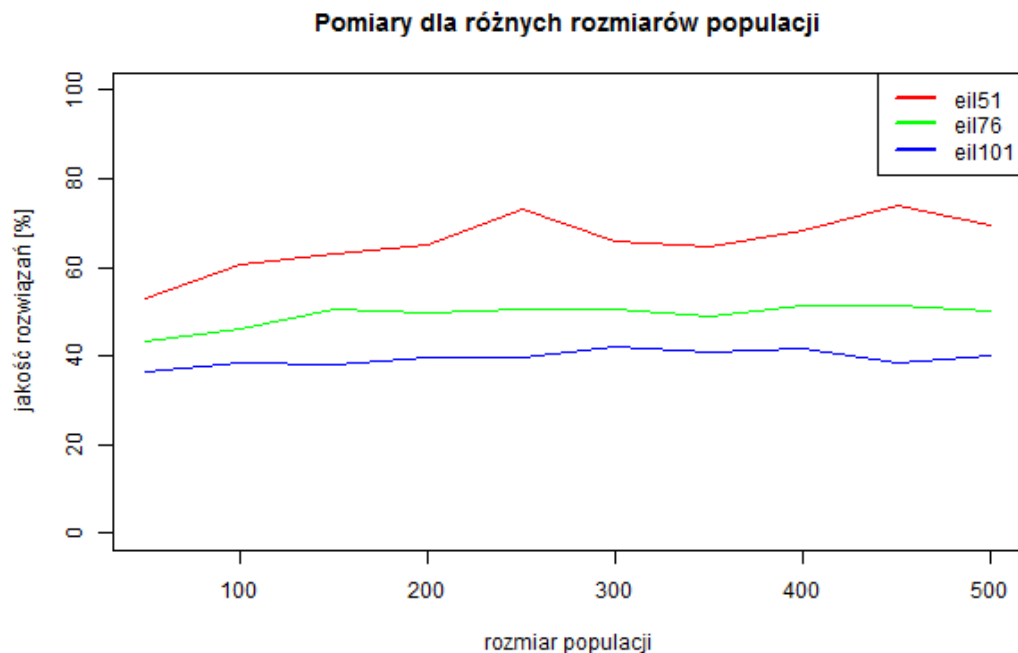
Przeprowadzono badania z zakresu optymalizacji marszruty dla problemu komiwojażera. Wykorzystano trzy instancje problemu z biblioteki TSPLIB:

- eil51
- eil76
- eil101

Jakość rozwiązań wyraża się wzorem:

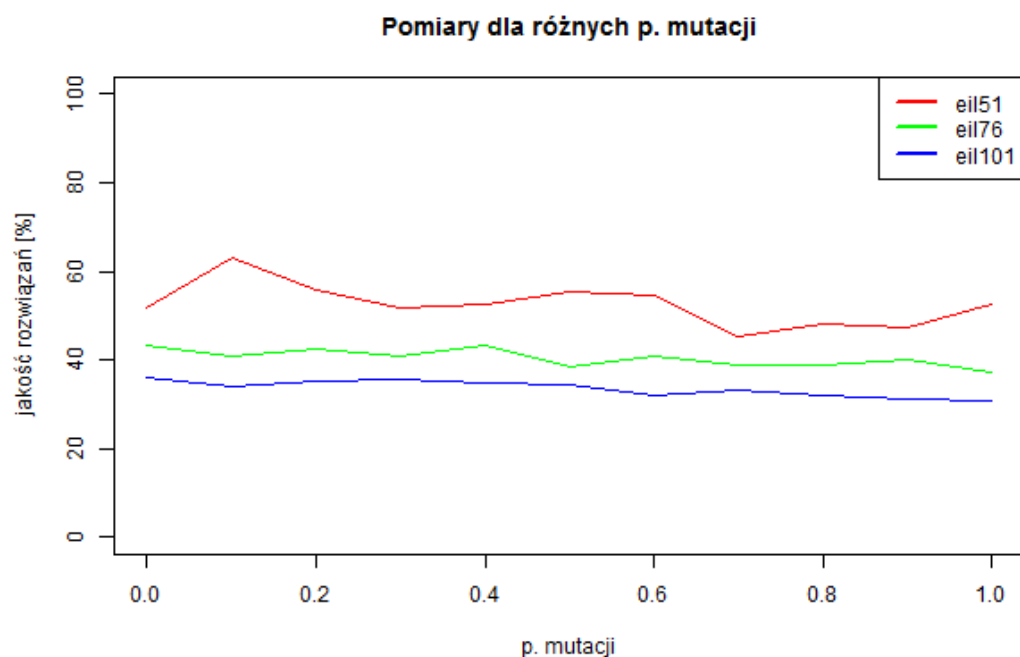
$$quality\ of\ solution = \frac{shortest\ known\ path}{found\ path} * 100\% \quad (2)$$

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



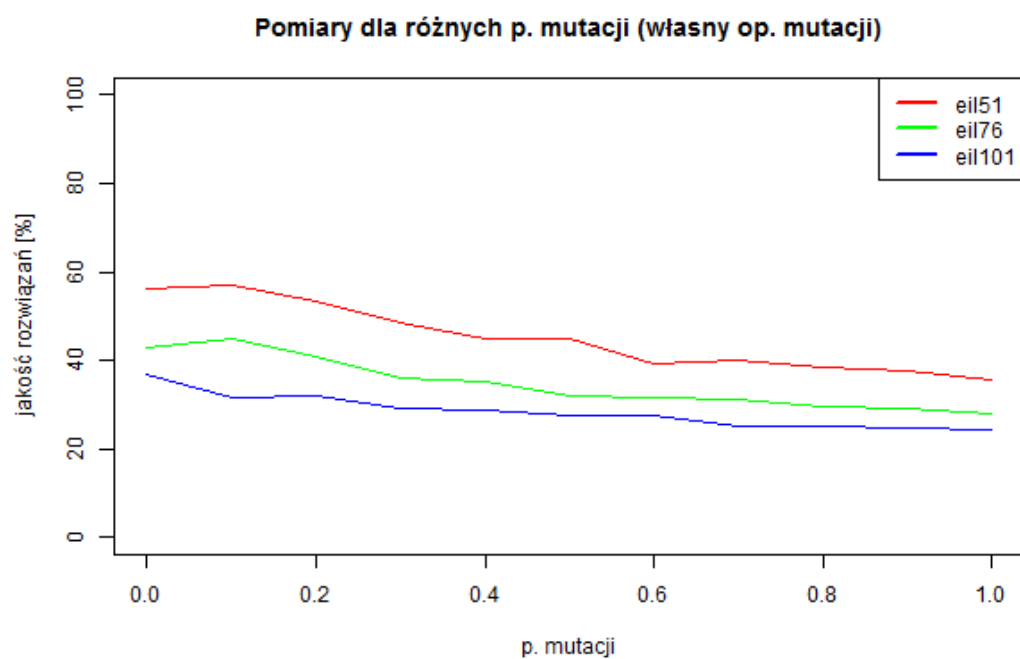
Rysunek 6: Jakość rozwiązań dla różnych rozmiarów populacji

Na ilustracji (rys. 7) 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 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)