

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	Problem optymalizacji rzeczywistej (funkcja Hartman6)	3
2.1	Badanie algorytmu genetycznego dla własnej funkcji mutacji	3
2.2	Badanie algorytmów dla różnych wartości ich unikalnych parametrów . . .	6
3	Problem komiwojażera	9
3.1	Przebieg i wyniki badań dla algorytmu genetycznego	9
3.2	Słowo na temat operatora mutacji	11
4	Implementacja	12
5	Podsumowanie	22

1 Wprowadzenie

Algorytm genetyczny jest algorytmem heurystycznym, który swoim działaniem przypomina działanie ewolucji w naturze. Osobniki będące zbyt słabymi zostają wyeliminowane z populacji w kolejnych pokoleniach, a na ich miejsce przyjmowane są lepsze, silniejsze, bardziej podatne adaptacji. Algorytm ten zakłada możliwość mutacji i krzyżowania wśród potomków, przez co nie zawsze są oni silniejsi od poprzednio wyeliminowanych członków. Dodatkowo wprowadza się pojęcie elity, która jest bezpośrednio przenoszona do następnego - teoretycznie lepszego pokolenia.

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

Algorytm PSO jest to natomiast algorytm optymalizacji rojem cząstek. Każda z cząstek z populacji (roju) jest umieszczana na losowej pozycji w przestrzeni rozwiązań. Pozycja każdej z cząstki podlega ocenie poprzez funkcję dopasowania. Z każdą cząstką związany jest dodatkowo pewien wektor prędkości z jaką się ona porusza. W każdej iteracji wektory odpowiadające pozycji cząstki oraz prędkości są sumowane. Na prędkość każdej z cząstek ma wpływ pozycja o największym dopasowaniu znaleziona przez cały rój oraz pozycja o największym dopasowaniu znaleziona przez konkretną cząstkę. Przebieg algorytmu dla przestrzeni 3-wymiarowej może być w interesujący sposób zwizualizowany.

Problem komiwojażera - jest zagadnieniem optymalizacyjnym, w którym optymalizowana jest długość trasy między miastami, tak by odnaleźć najkrótszą drogę nie omijając żadnego z nich. W części poświęconej temu problemowi zamieszczono szersze wyjaśnienie.

W ramach laboratorium należało przeprowadzić testy algorytmów porównując działanie domyślnej i własnej funkcji mutacji (zarówno dla problemu optymalizacji rzeczywistej jak i TSP). Dodatkowo należało wykonać testy unikalnych parametrów algorytmu memetycznego i PSO.

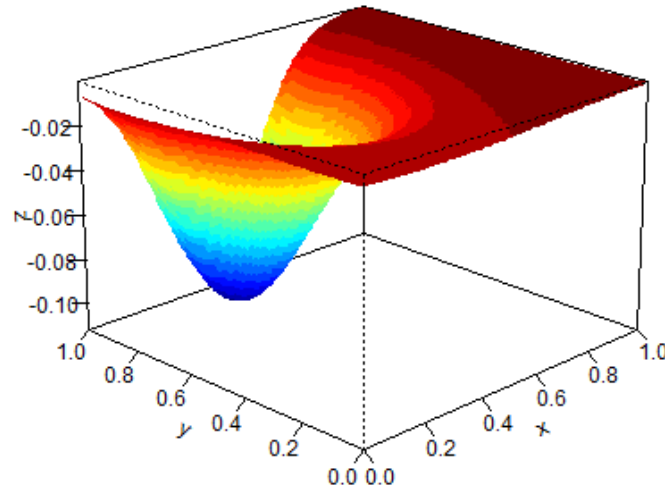
Pomiary wykonywano na 2 różnych jednostkach roboczych. Ich parametry nie są istotne z punktu widzenia analizy i możliwości porównania rezultatów.

2 Problem optymalizacji rzeczywistej (funkcja Hartman6)

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

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

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

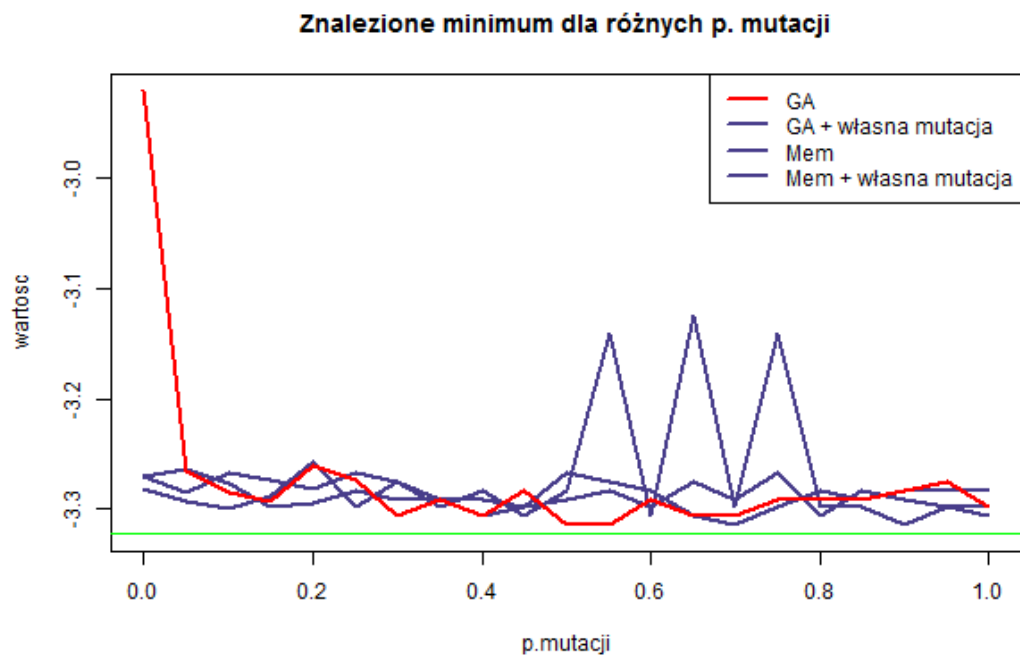


Rysunek 1: Wykres funkcji Hartman6

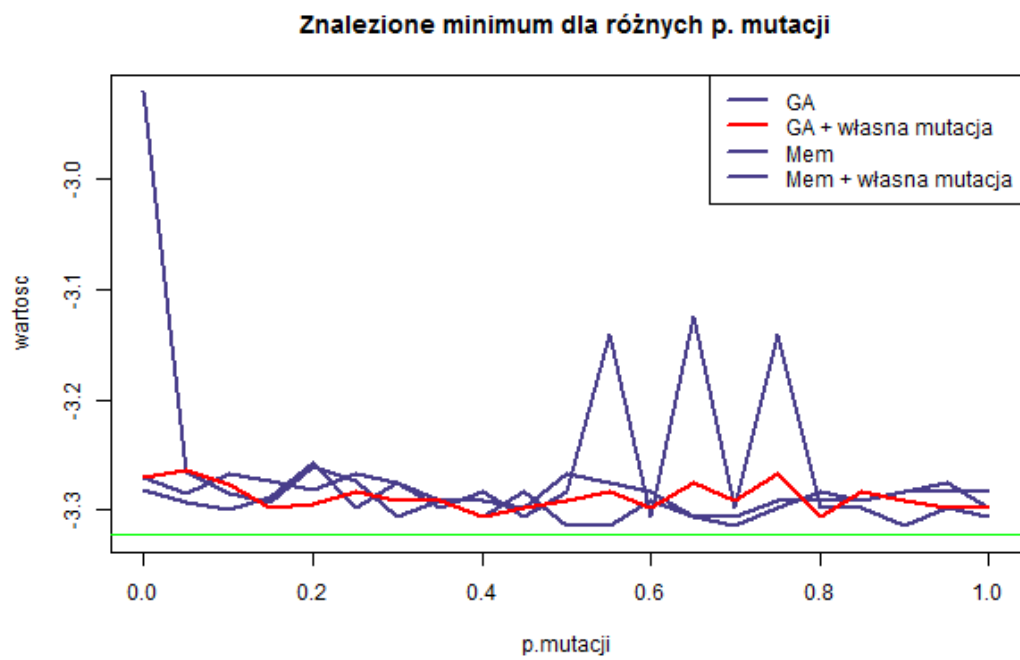
2.1 Badanie algorytmu genetycznego dla własnej funkcji mutacji

Badania przeprowadzono dla algorytmu genetycznego w wersji podstawowej, ze zmienioną funkcją mutacji oraz hybrydowej, a także dla algorytmu optymalizacji rojem cząstek (PSO). W tym punkcie zamieszczono wyniki dotyczące wpływu własnej funkcji mutacji na poziom optymalizacji.

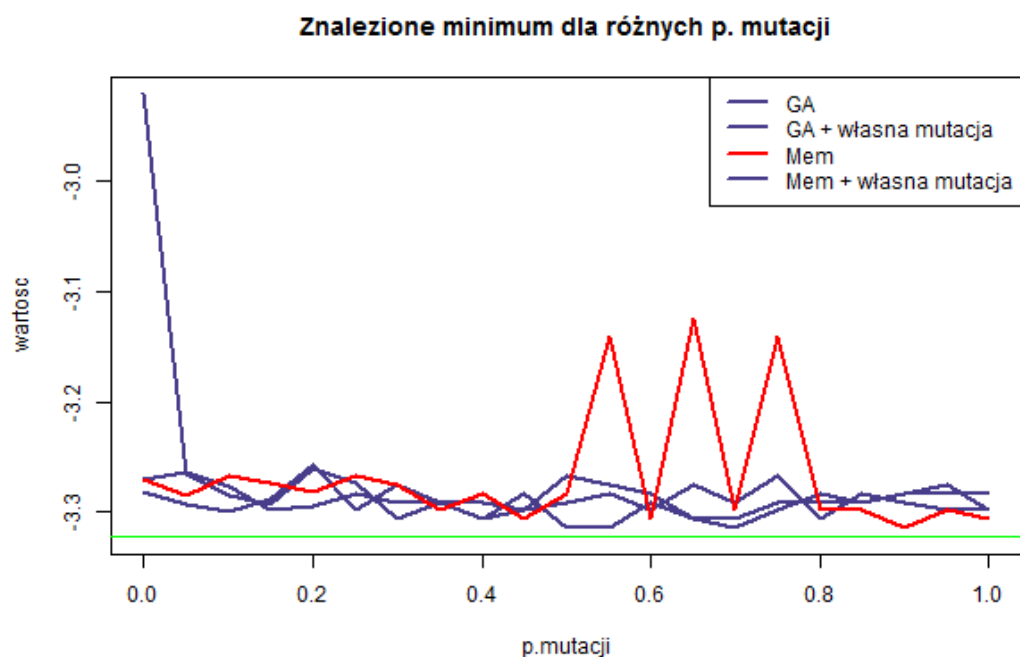
Własna funkcja mutacji została utworzona w taki sposób by nie doprowadzić do sytuacji, w której przekroczona zostanie minimalna lub maksymalna wartość populacji. Jej działanie opiera się na wybraniu minimalnej jednostki z populacji i podmianie innej, losowej na znalezioną minimalną. Gwarantuje to niepojawienie się w populacji wartości uznawanej przez algorytm za niepoprawną.



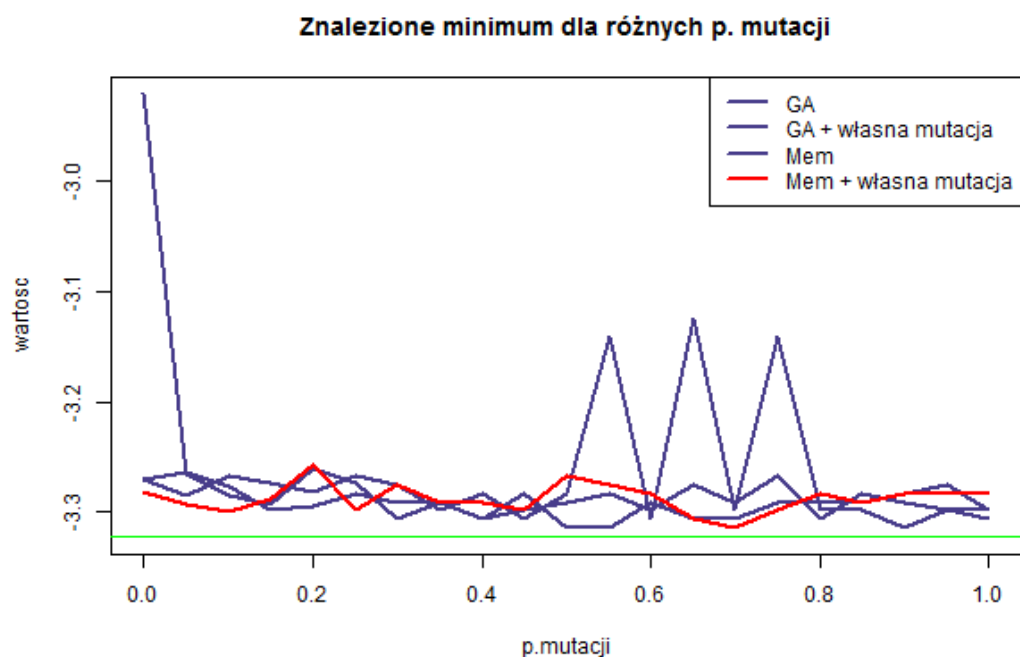
Rysunek 2: Wykres funkcji Hartman6



Rysunek 3: Wykres funkcji Hartman6



Rysunek 4: Wykres funkcji Hartman6



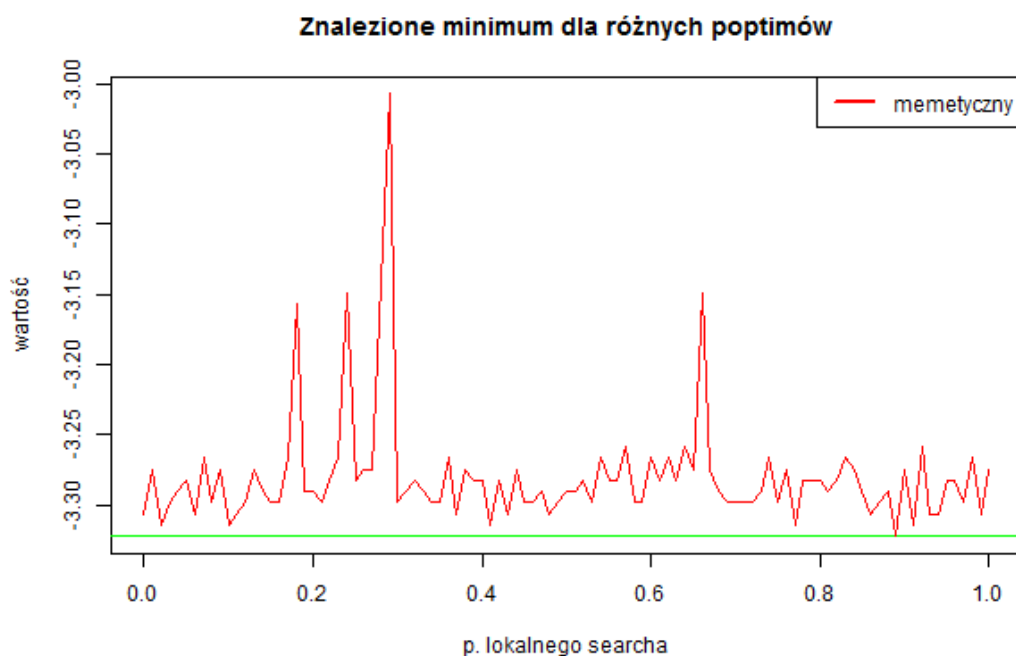
Rysunek 5: Wykres funkcji Hartman6

Z wykresów można odczytać zbliżone wyniki dla różnych funkcji mutacji. Żadna z funkcji nie osiągnęła minimum co świadczy o tym, że sama mutacja tutaj nie jest wystarczająca. 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 widocznie odstające od reszty.

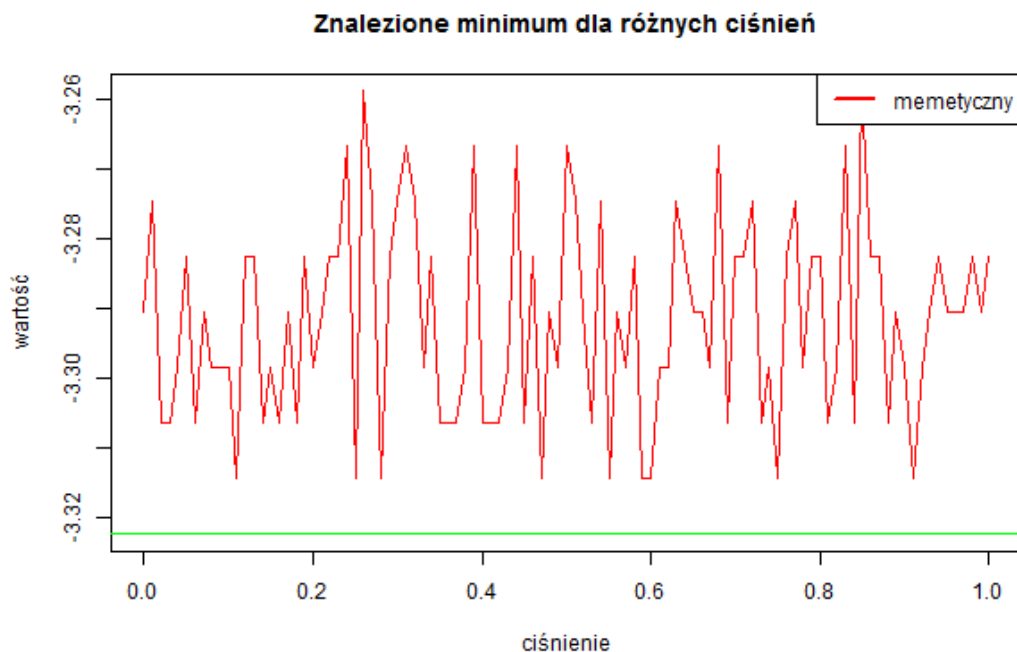
2.2 Badanie algorytmów dla różnych wartości ich unikalnych parametrów

Algorytmy memetyczny i PSO posiadają własne wartości unikalne, których zmiana może wpłynąć na otrzymywane wyniki. Poniżej przedstawiono wykresy przedstawiające otrzymane wartości optimum przy zmianie wybranych parametrów. Badania przeprowadzono w przedziale 0 – 1 z krokiem co 0,01. Otrzymane wyniki są uśrednione na podstawie 30 iteracji.



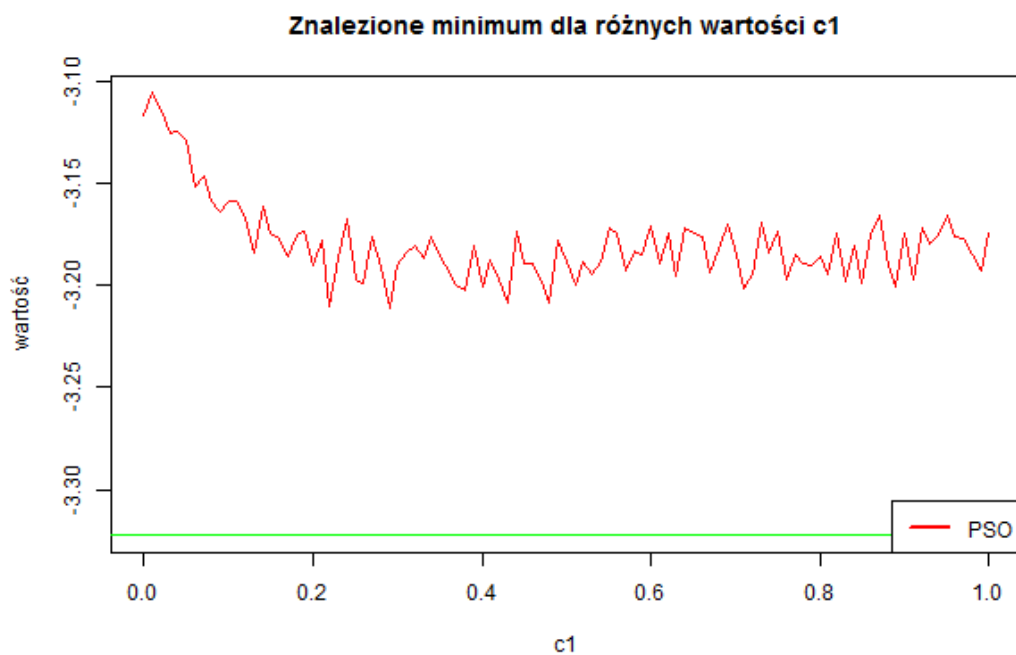
Rysunek 6: Jakość rozwiązań dla różnych wartości poptimum (memetyczny)

Z wykresu (rys. 6) można odczytać niski wpływ wartości poptimum na otrzymane wyniki. Otrzymywane wartości różnią się nie więcej niż o 0,30. Przy czym całość zdaje się mieć charakter mocno nieuporządkowany, w związku z czym nie można określić optymalnej wartości parametru. Zauważalne są 4 „skoki” zawyżające skalę rezultatów spowodowane jednak najprawdopodobniej specyfiką całego algorytmu – nie można się tu doszukiwać prawidłowości.



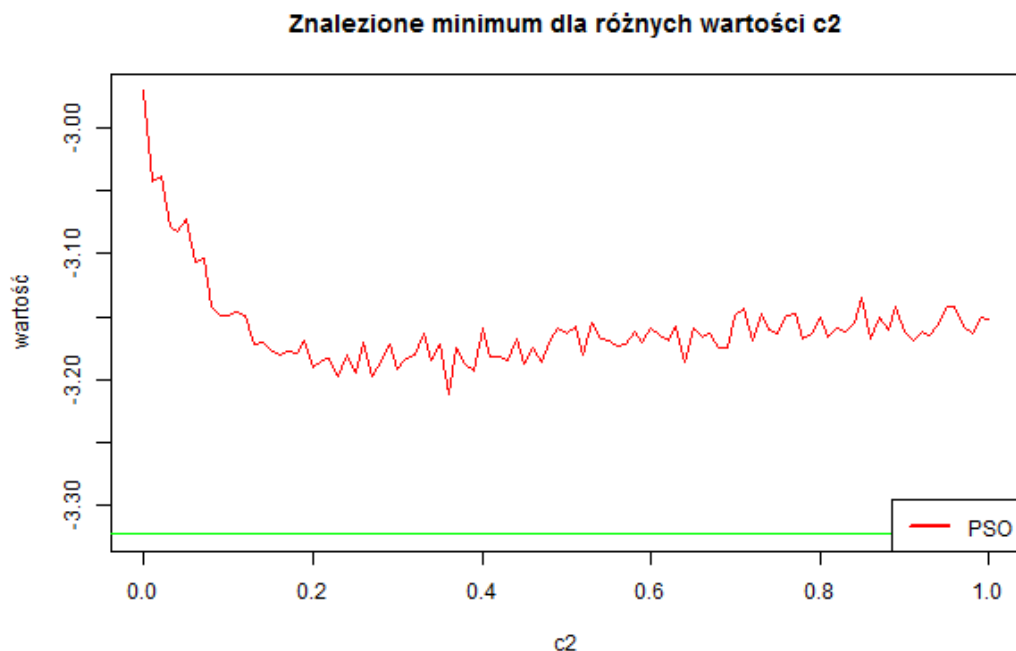
Rysunek 7: Jakość rozwiązań dla różnych wartości ciśnienia (memetyczny)

Wartości na wykresie (rys. 7) różnią się od siebie o nie więcej niż 0,06. Oznacza to niski wpływ ciśnienia na rezultat algorytmu podobnie jak w przypadku lokalnego wyszukiwania poprzednio. Nie zauważalne są trendy w zakresie wartości wraz ze wzrostem ciśnienia. Nie można wskazać zalecanej wartości parametru na podstawie przeprowadzonego badania.



Rysunek 8: Jakość rozwiązań dla różnych wartości parametru $c1$ algorytmu PSO

Z przeprowadzonych badań wynika, że najlepsze rezultaty otrzymano dla wartości $c1$ przekraczających 0,2. Natomiast po przekroczeniu tego umownego progu różnice wyników spowodowane są już tylko heurystyką algorytmu. Ogólnie rzecz biorąc gdyby należało wskazać zalecany parametr domyślny to byłoby to 0,2 na podstawie tych pomiarów.



Rysunek 9: Jakość rozwiązań dla różnych wartości parametru $c2$ algorytmu PSO

Tak jak w przypadku parametru $c1$ najlepsze rezultaty uzyskano po przekroczeniu wartości 0,2. Jednak w tym przypadku można zauważyć powolne pogorszenia się wyników wraz ze wzrostem wartości parametru $c2$. Zatem podobnie jak poprzednio sugerowaną wartością domyślną powinno być 0,2 – chociaż być może tylko dla tej konkretnie badanej funkcji.

Algorytm PSO badano dla ilości cząstek równej 500 i ilości przebiegów równej 50.

3 Problem komiwojażera

Problem komiwojażera (ang. travelling salesman problem) w wersji optymalizacyjnej polega na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym, który ma n wierzchołków – można przyjąć, że każdy wierzchołek reprezentuje miasto. Waga każdej krawędzi może oznaczać odległość pomiędzy konkretnymi miastami. Rozróżnić można wersję symetryczną (drogi z miasta A do miasta B oraz z miasta B do A mają jednakowe wagi) bądź niesymetryczną, w której odległości między miastami różnią się w zależności od punktu startowego. Problem sprowadza się do znalezienia takiej sekwencji odwiedzania miast, by każde z nich zostało odwiedzone tylko raz, a sumaryczna droga (a więc suma wag krawędzi) była jak najmniejsza.

Parametrami zadania są:

- skończony zbiór n -miast $C = c_1, c_2, \dots, c_n$
- skończony zbiór odległości d_{ij} , z miasta c_i do c_j
- jeśli rozpatrujemy wersję symetryczną $d_{ij} = d_{ji}$
- jeśli rozpatrujemy wersję asymetryczną, brak powyższego wymogu

Należy określić kolejność odwiedzania wszystkich miast $\langle c_{i1}, c_{i2}, \dots, c_{in} \rangle$, aby sumaryczna trasa była jak najkrótsza, przy założeniu, że każde miasto zostało odwiedzone dokładnie jeden raz.

$$\min\left(\sum_{j=1}^{n-1} d_{ij, i(j+1)} + d_{in, i1}\right) \quad (2)$$

3.1 Przebieg i wyniki badań dla algorytmu genetycznego

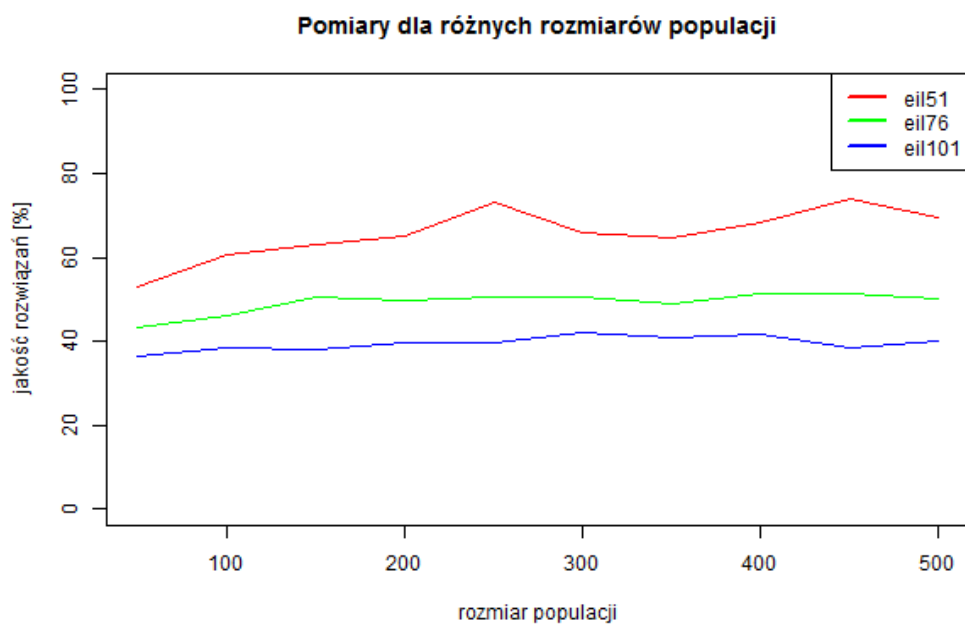
Badania z zakresu optymalizacji marszruty przeprowadzono dla symetrycznego problemu komiwojażera z wagami będącymi odległościami euklidesowymi. Wykorzystano trzy instancje z biblioteki TSPLIB:

- eil51
- eil76
- eil101

W celu porównania osiągnięć algorytmu pomiędzy instancjami wprowadzono metrykę jakości rozwiązań wyrażoną wzorem:

$$\text{quality of solution} = \frac{\text{shortest known path}}{\text{best found path}} * 100\% \quad (3)$$

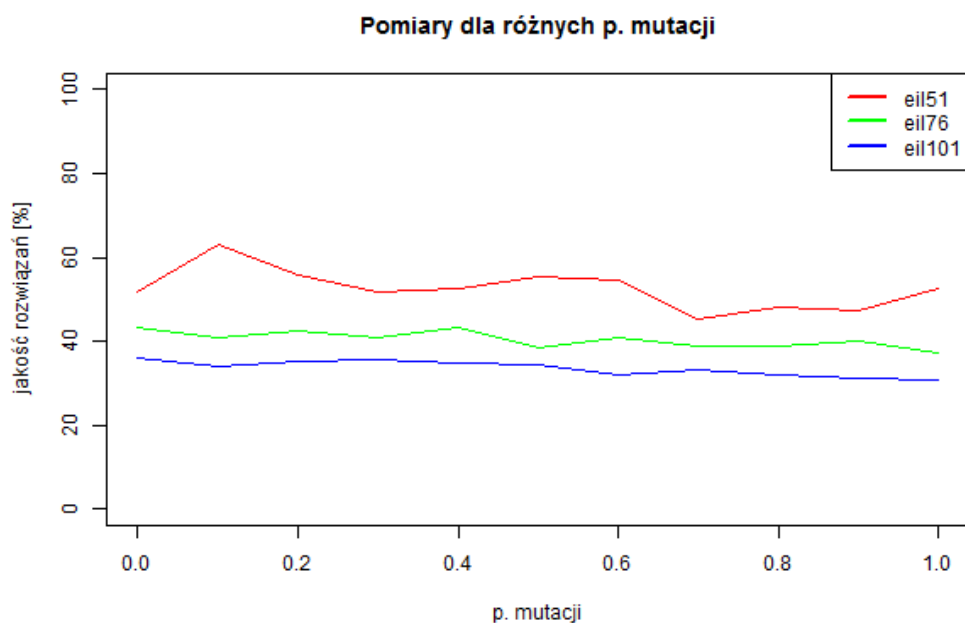
Wszystkie pomiary powtórzone zostały 15-krotnie celem uśrednienia. Na ilustracji (rys. 10) przedstawiono wyniki dla różnych rozmiarów populacji.



Rysunek 10: 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. Natomiast ogólnie rzecz biorąc wyniki są dalekie od ideału.

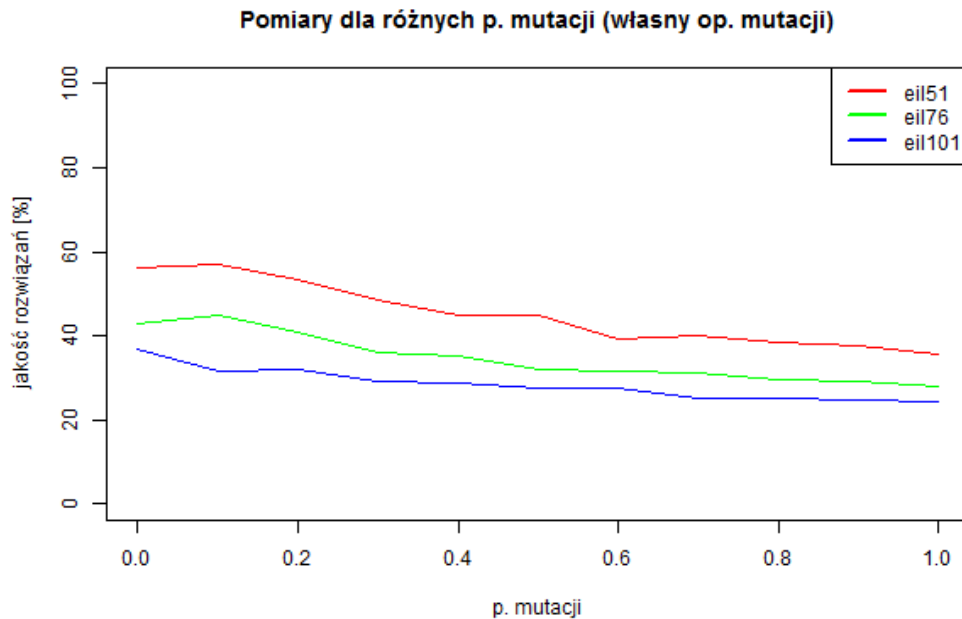
Na ilustracji (rys. 11) przedstawiono wyniki pomiarów dla różnych wartości p . mutacji dla domyślnego operatora.



Rysunek 11: Jakość rozwiązań dla różnych wartości p . mutacji

Na podstawie powyższego wykresu nie można ustalić bezpośredniego wpływu prawdopodobieństwa mutacji na jakość rozwiązań dla instancji „eli76” i „eli101”. Dla instancji „eli51” jakość wydaje się zmniejszać wraz ze wzrostem prawdopodobieństwa mutacji. Jednakże nie są to różnice, które pozwalają na wskazanie optymalnej wartości dla omawianego parametru.

Na ilustracji (rys. 12) przedstawiono wyniki pomiarów dla różnych wartości p . mutacji z niestandardowym operatorem.



Rysunek 12: 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.

3.2 Słowo na temat operatora mutacji

Operator mutacji otrzymano poprzez połączenie dostępnych w pakiecie *GA* operatorów odpowiednio: insertion (losowo wybrane miasto jest usuwane i wstawiane ponownie do sekwencji w inne losowe miejsce), displacement (miasta z wybranego losowo zakresu są przemieszczane w inne miejsce bez zmiany kolejności sekwencji) oraz scramble (losowo wybierany jest zakres pomiędzy którym kolejność miast jest mieszana).

4 Implementacja

Poniżej zamieszczono kody skryptów w języku R przygotowanych w celu umożliwienia przeprowadzenia pomiarów. Wykorzystano dwa osobne skrypty kolejno dla optymalizacji funkcji z pakietu „globalOptTests” i problemu komiwojażera.

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 <- 30 # 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
            gaControl("real-valued")$mutation,
92           fitness = function(xx) -f(xx),
93           min = c(B[1,]), max = c(B[2,]),
94           popSize = if (mType == "pop") value else 50,
95           pmutation = if (mType == "mut") value else 0.1)
96         solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
97         eval <- f(solution[1,])
98         sum <- sum + eval
99       }

```

```

100     averages <- c(averages, (sum / nOfRuns))
101   }
102   temp <- c(temp, averages)
103 }
104 result <- matrix(c(temp), nrow = length(series), ncol = length(values))
105
106 if (graphs) {
107
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                    popsize = 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",
187                     mutation = if (serie == 2) myMutationFunction else
188                               gaControl("real-valued")$mutation,
189                     fitness = function(xx) -f(xx),
190                     min = c(B[1,]), max = c(B[2,]),
191                     popSize = if (mType == "pop") value else 50,
192                     pmutation = if (mType == "mut") value else 0.1)
193         }
194         else
195         {
196           GAmin <- ga(type = "real-valued",
197                     mutation = if (serie == 4) myMutationFunction else
198                               gaControl("real-valued")$mutation,
199                     fitness = function(xx) -f(xx),
200                     min = c(B[1,]), max = c(B[2,]),
201                     optim = TRUE,
202                     optimArgs = list (
203                       poptim = if (mType == "poptim") value else 0.05,
204                       pressel = if (mType == "pressel") value else 0.5))
205         }
206
207         solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
208         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,
295        height=400, units="px")
296   plot(0, 0, main=title,
297        ylim=c(min(c(averages,globalOpt)),max(c(averages,globalOpt))),
298        xlim=c(min(values),max(values)),
299        type="n", xlab=xLabel, ylab="wartość")
300   abline(globalOpt,0, col="green")
301   lines(values, averages, col = "red", type = 'l')
302   legend("bottomright", c("PSO"), lwd=rep(2,1), lty=rep(1,1), col=c("red"))
303   dev.off()
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. Poniżej pokrótce omówiono podstawowe parametry.

- nOfRuns
Ilość powtórzeń dla każdego pomiaru w celu uśrednienia.
- colors, GAWithHybridColors, series, GAWithHybridSeries
Wektory kolorów i nazw kolejnych serii pomiarowych dla różnych rodzajów pomiarów.
- funcName
Nazwa funkcji dla których przeprowadzane są pomiary.
- poptim, pressel
Domyślne 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 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 badania algorytmu genetycznego 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 <- 15
13
14 # instances for testing 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   # insertion mutation
29   parent <- as.vector(object@population[parent,])
30   n <- length(parent)
31   m <- sample(1:n, size = 1)
32   pos <- sample(1:(n-1), size = 1)
33   i <- c(setdiff(1:pos,m), m, setdiff((pos+1):n,m))
34   mutate <- parent[i]
35
36   # displacement mutation
37   parent <- mutate
38   m <- sort(sample(1:n, size = 2))
39   m <- seq(m[1], m[2], by = 1)
40   l <- max(m)-min(m)+1
41   pos <- sample(1:max(1,(n-1)), size = 1)
42   i <- c(setdiff(1:n,m)[1:pos], m, setdiff(1:n,m)[- (1:pos)])
43   mutate <- parent[na.omit(i)]
44
45   # scramble mutation
46   parent <- mutate
47   m <- sort(sample(1:n, size = 2))
48   m <- seq(min(m), max(m), by = 1)
49   m <- sample(m, replace = FALSE)
50   i <- c(setdiff(1:min(m),m), m, setdiff(max(m):n,m))
51   mutate <- parent[i]
52   return(mutate)
53 }
54

```

```

55 performTest <- function(testName, graphMain, graphXLab,
56                         sequenceType, sequence,
57                         popsize=50, pcrossover=0.8,
58                         pmutation=0.1, maxiter=100, mutation = NULL) {
59
60     solution_qualities <- c()
61
62     # each instance as separate serie
63     for (i in 1:length(instances)) {
64         fileName = paste("examples/", instances[i], ".tsp", sep="")
65         graphTitle = paste("TSPLIB: ", instances[i], sep="")
66
67         drill <- read_TSPLIB(system.file(fileName, package = "TSP"))
68         D <- as.matrix(dist(drill, method = "euclidean"))
69         N <- max(dim(D))
70
71         solution_quality <- c()
72         bestTour <- NA
73         bestTourLength <- .Machine$integer.max
74         averageLength <- 0
75         for (s in 1:length(sequence)) {
76             for (n in 1:numberOfMeasurements) {
77                 message(paste("Instancja: ", i))
78                 message(paste("Sekwencja: ", s))
79                 message(paste("Pomiar: ", n))
80
81                 GA <- ga(type = "permutation",
82                        fitness = fit,
83                        distMatrix = D,
84                        min = 1,
85                        max = N,
86                        popSize = if (sequenceType == "popsize") sequence[s] else
87                                popsize,
88                        pcrossover = if (sequenceType == "pcrossover") sequence[s] else
89                                pcrossover,
90                        pmutation = if (sequenceType == "pmutation") sequence[s] else
91                                pmutation,
92                        maxiter = if (sequenceType == "maxiter") sequence[s] else
93                                maxiter,
94                        mutation = if (is.null(mutation))
95                                gaControl("permutation")$mutation else mutation)
96
97                 tour <- GA@solution[1, ]
98                 tl <- tourLength(tour, D)
99                 if (tl < bestTourLength) {
100                     bestTourLength <- tl
101                     bestTour <- tour
102                 }
103                 averageLength <- averageLength + (tl - averageLength) / n
104             }
105             solution_quality <- c(solution_quality,
106                                (best_solutions[i]/averageLength) * 100)
107         }
108     }
109
110     png(file = paste(testName, "_", instances[i], ".png", sep=""), width=600,
111         height=400, units="px")

```

```

105 plot(drill, bestTour, cex=.6, col = "red", pch=3, main = graphTitle)
106 dev.off()
107
108 solution_qualities <- c(solution_qualities, solution_quality)
109 }
110
111 qualities = matrix(solution_qualities, nrow=length(instances),
112                    ncol=length(sequence), byrow = TRUE)
113 # save graph with measurement series to file
114 png(file = paste(testName, ".png", sep=""), width=600, height=400, units="px")
115 plot(0, 0, main=graphMain,
116      ylim=c(0,100),
117      xlim=c(min(sequence),max(sequence)),
118      type="n", xlab=graphXLab, ylab="jakość rozwiązań [%]")
119 for (i in 1:length(instances)) {
120   lines(sequence, qualities[i,], col = colors[i], type = 'l')
121 }
122 legend("topright", instances, lwd=rep(2,length(instances)),
123       lty=rep(1,length(instances)), col=colors)
124 dev.off()
125 }
126
127 performTest(testName = "tsp_pop", graphMain = "Pomiary dla różnych rozmiarów
128          populacji",
129            graphXLab = "rozmiar populacji",
130            sequenceType = "popsize", sequence = seq(50, 500, 50))
131
132 performTest(testName = "tsp_mut", graphMain = "Pomiary dla różnych p. mutacji",
133            graphXLab = "p. mutacji",
134            sequenceType = "pmutation", sequence = seq(0, 1, 0.1))
135
136 performTest(testName = "tsp_mut_custom", graphMain = "Pomiary dla różnych p.
137          mutacji (własny op. mutacji)",
138            graphXLab = "p. mutacji",
139            sequenceType = "pmutation", sequence = seq(0, 1, 0.1), mutation =
140            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.

5 Podsumowanie

W trakcie przeprowadzonych badań przetestowano algorytmy w wariantach: genetyczny domyślny, genetyczny z własną funkcją mutacji, hybrydowy domyślny oraz hybrydowy z własną funkcją mutacji a także algorytm optymalizacji rojem cząstek – dla problemu optymalizacji rzeczywistej. Przeprowadzono także testy dla algorytmu genetycznego (domyślnego jak również ze zmodyfikowaną funkcją mutacji) dla problemu TSP.

Zmiana funkcji mutacji dla optymalizacji rzeczywistej nie spowodowała znaczących różnic w jakości otrzymywanych rozwiązań. Jej działanie okazało się porównywalne z zaimplementowaną funkcją. Przygotowana funkcja mutacji będąca złożeniem kilku domyślnych funkcji mutacji okazała się gorsza od domyślnej (pogorszyła rezultaty o ok. 10 punktów procentowych).

Z przeprowadzonych badań wynika brak lub niski wpływ dokonanych modyfikacji funkcji mutacji na działanie badanych algorytmów. Warto wspomnieć, że również w samym pakiecie GA jest możliwość wyboru spośród kilku dostępnych funkcji mutacji i krzyżowania.

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)