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 Poblem 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.

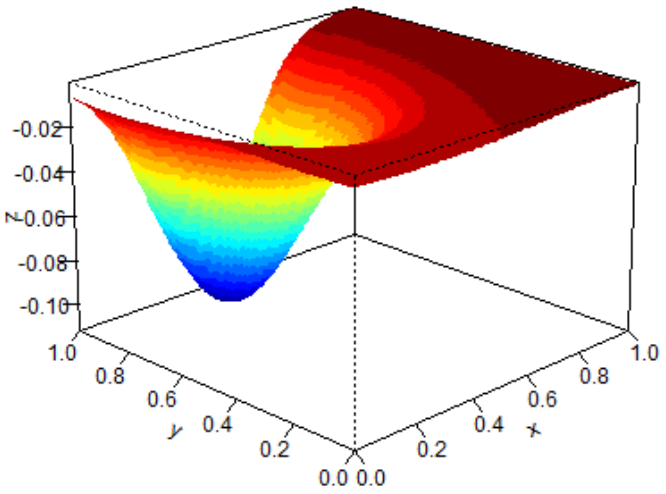
Pomiary wykonywano na 2 różnych jednostkach roboczych. Ich parametry nie są istotne z punktu widzenia analizy i możliwości porównania rezultatów.

2 Poblem 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).

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

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

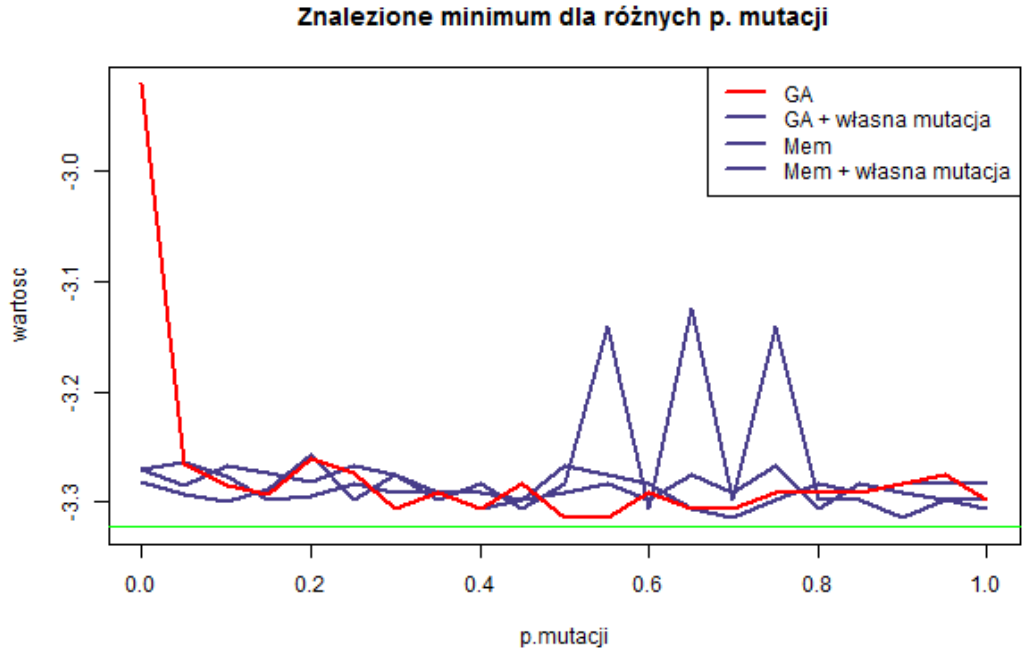


Rysunek 1: Wykres funkcji Hartman6

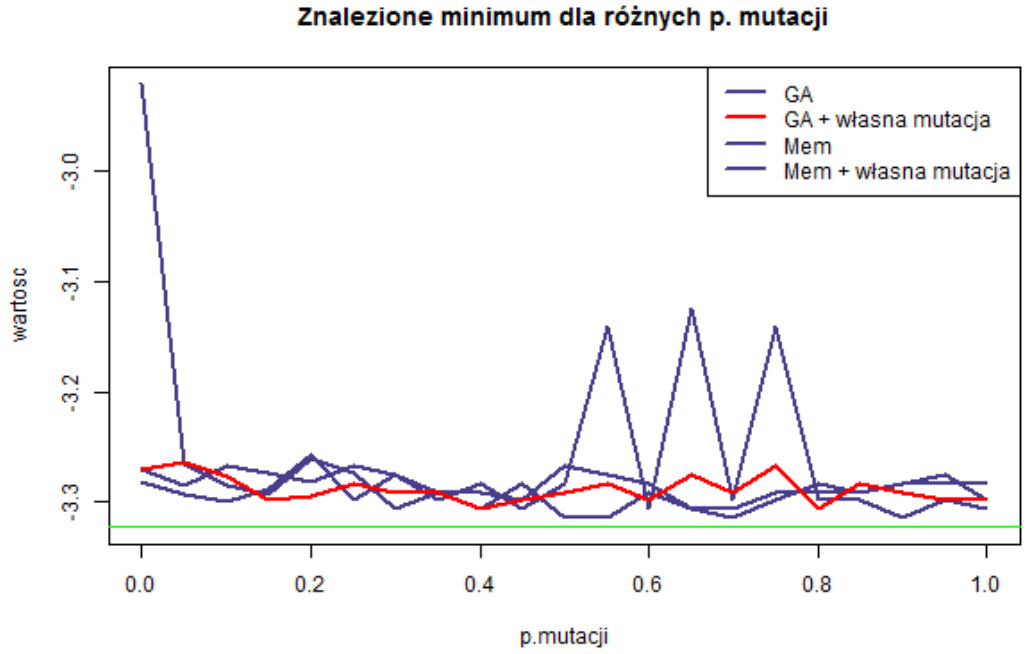
2.1 Badanie algorytmu genetycznego dla własnej funkcji mutacji

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.

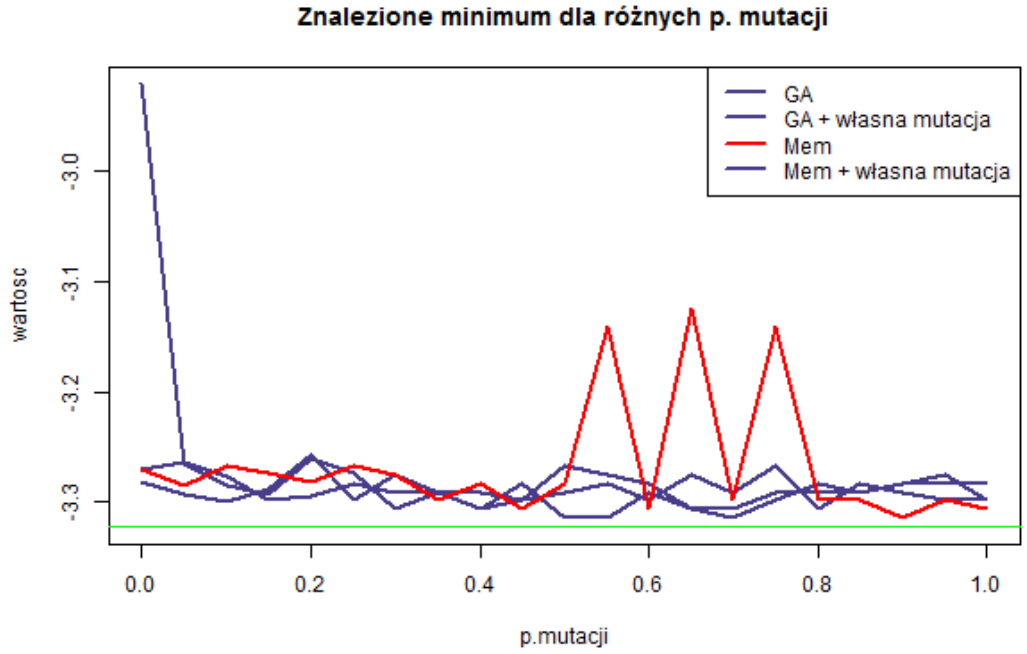
Własna funkcja mutacji została utworzona w taki sposób by nie doprowadzić do sytuacji, w której przekroczona zostanie minimalna lub maksymalna wartość populacji. Jej działanie opiera się na wybraniu minimalnej jednostki z populacji i podmianie innej, losowej na znalezioną minimalną. Gwarantuje to niepojawienie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



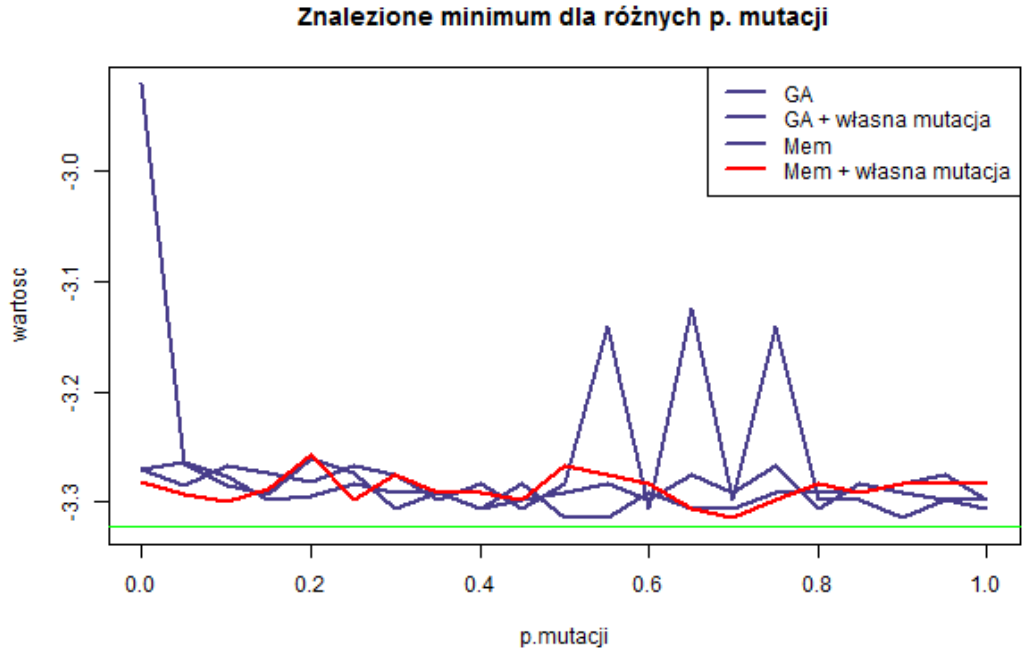
Rysunek 2: Wykres funkcji Hartman6



Rysunek 3: Wykres funkcji Hartman6



Rysunek 4: Wykres funkcji Hartman6



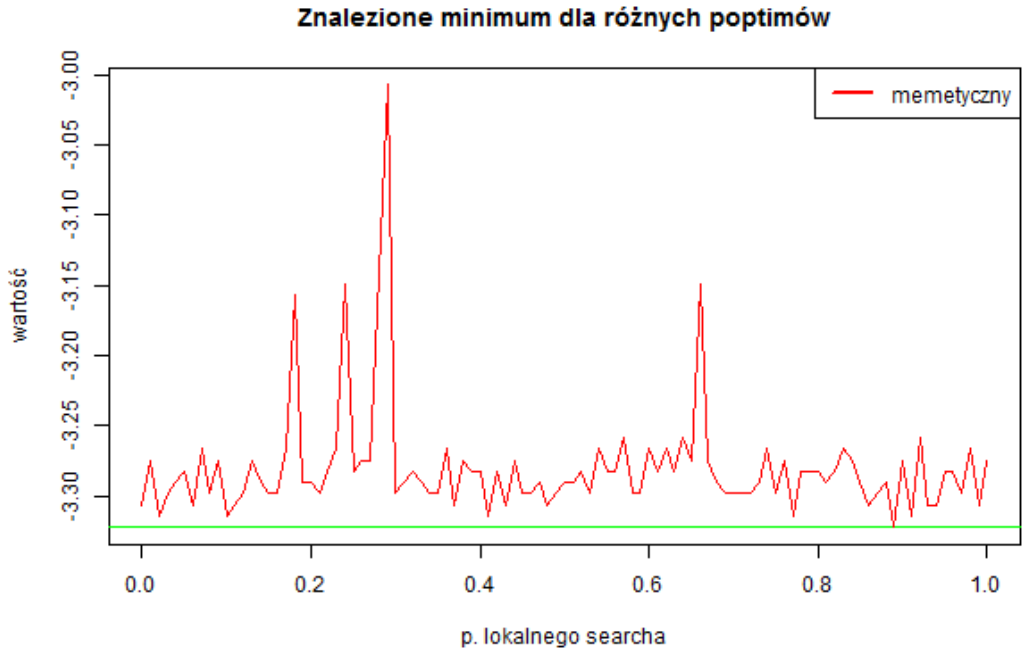
Rysunek 5: Wykres funkcji Hartman6

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.

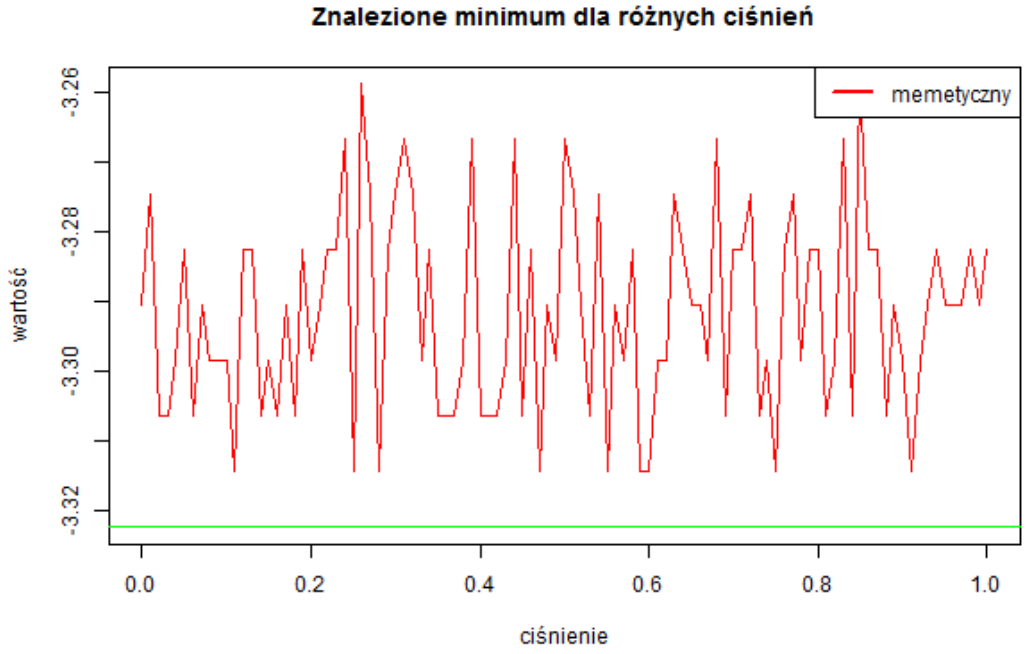
2.2 Badanie algorytmów dla różnych wartości ich unikalnych parametrów

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.



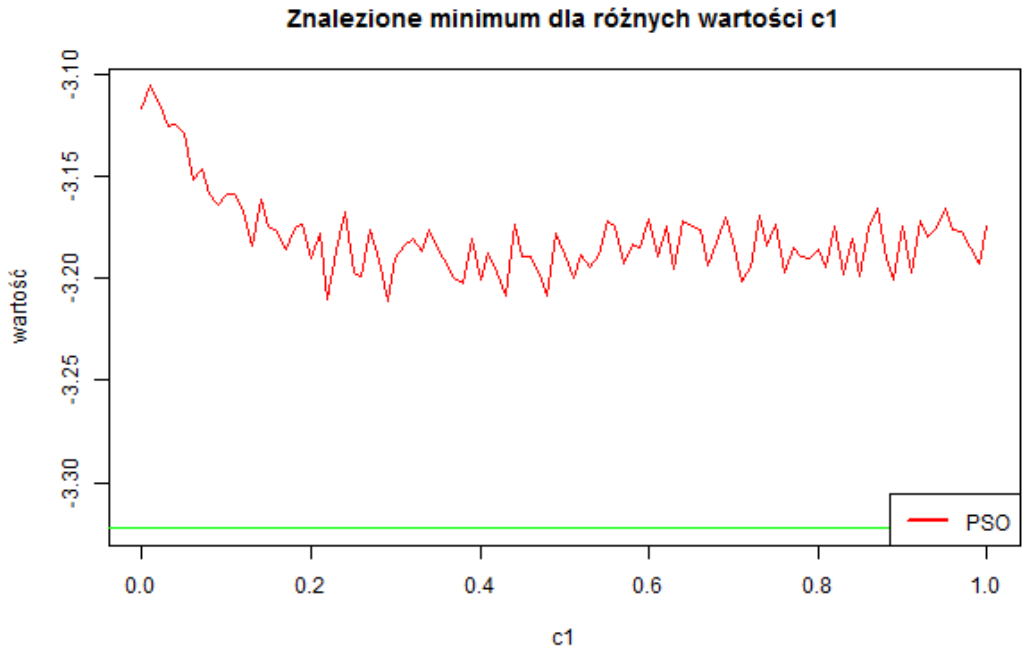
Rysunek 6: Jakość rozwiązań dla różnych wartości poptimum (memetyczny)

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.



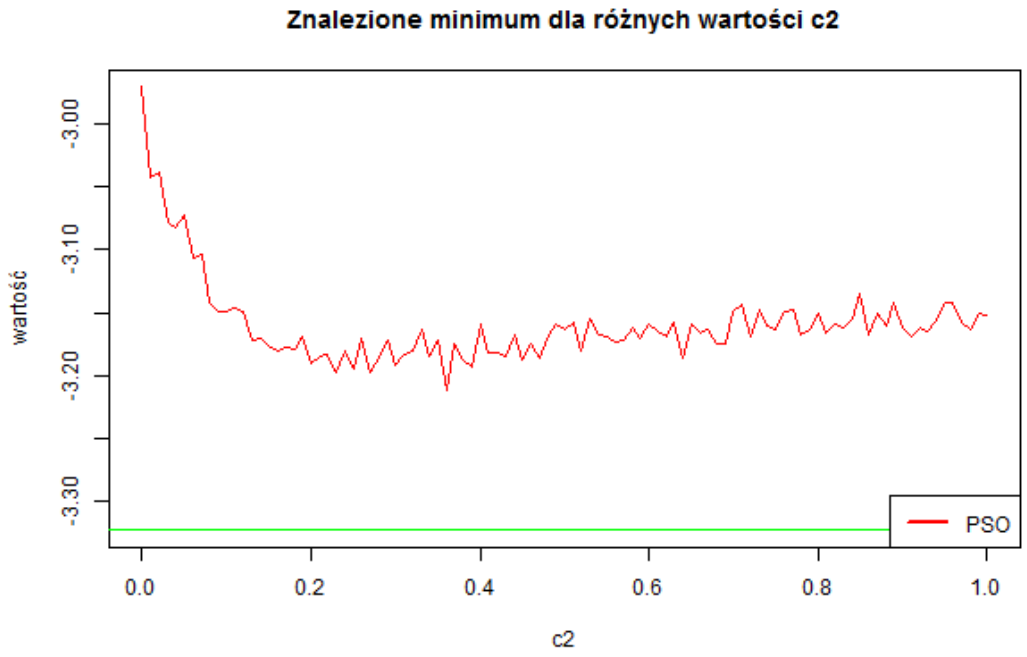
Rysunek 7: Jakość rozwiązań dla różnych wartości ciśnienia (memetyczny)

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 została odwiedzone tylko raz, a sumaryczna droga (a więc suma wag krawędzi) była jak najmniejsza.

Parametrami zadania są:

- skończony zbiór n-miast $C = c_1, c_2, \ldots, c_n$
- 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 $< c_{i1}, c_{i2}, ..., c_{in} >$, aby sumaryczna trasa była jak najkrótsza, przy założeniu, że każde miasto zostało odwiedzone dokładnie jeden raz.

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

3.1 Przebieg i wyniki badań dla algorytmu genetycznego

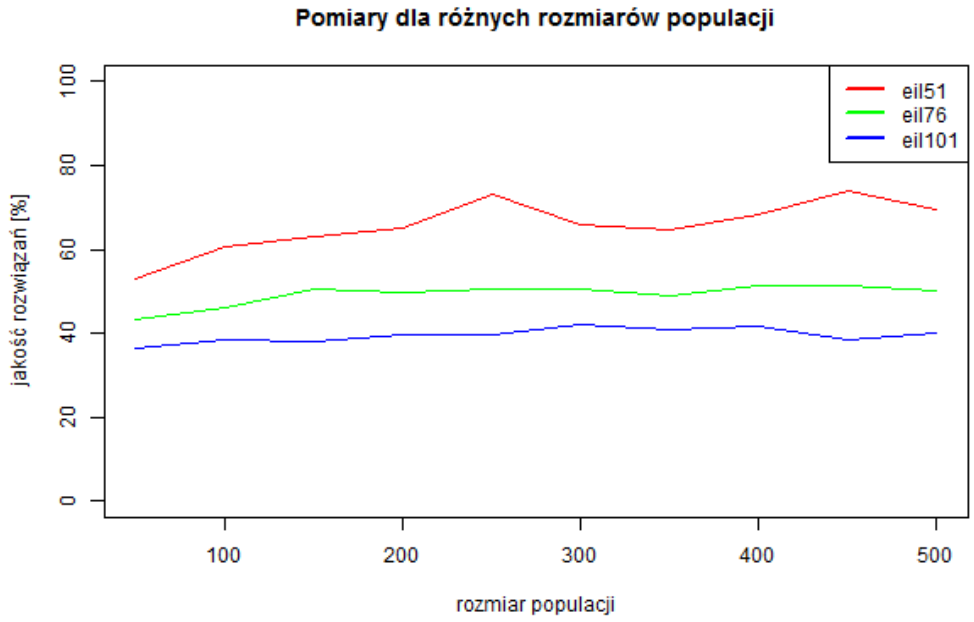
Badania z zakresu optymalizacji marszruty przeprowadzono dla symetrycznego problemu komiwojażera z wagami będącymi odległościami euklidesowymi. Wykorzystano trzy instancje z biblioteki TSPLIB:

- eil51
- eil76
- eil101

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

$$quality\ of\ solution = \frac{shortest\ known\ path}{best\ found\ path} * 100\% \tag{3}$$

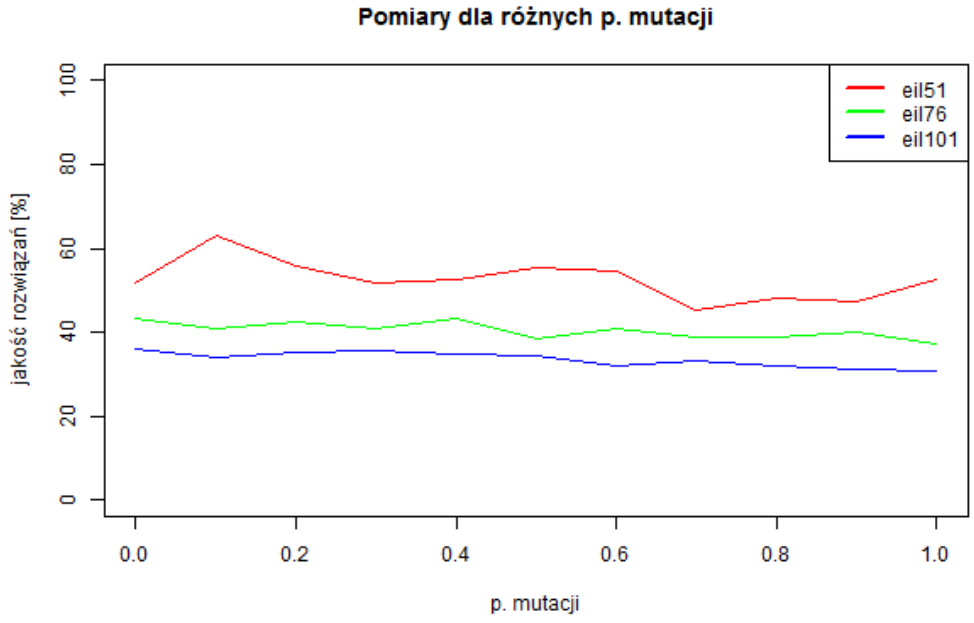
Wszystkie pomiary powtórzone zostały 15-krotnie celem uśrednienia. Na ilustracji (rys. 10) przedstawiono wyniki dla różnych rozmiarów populacji.



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.

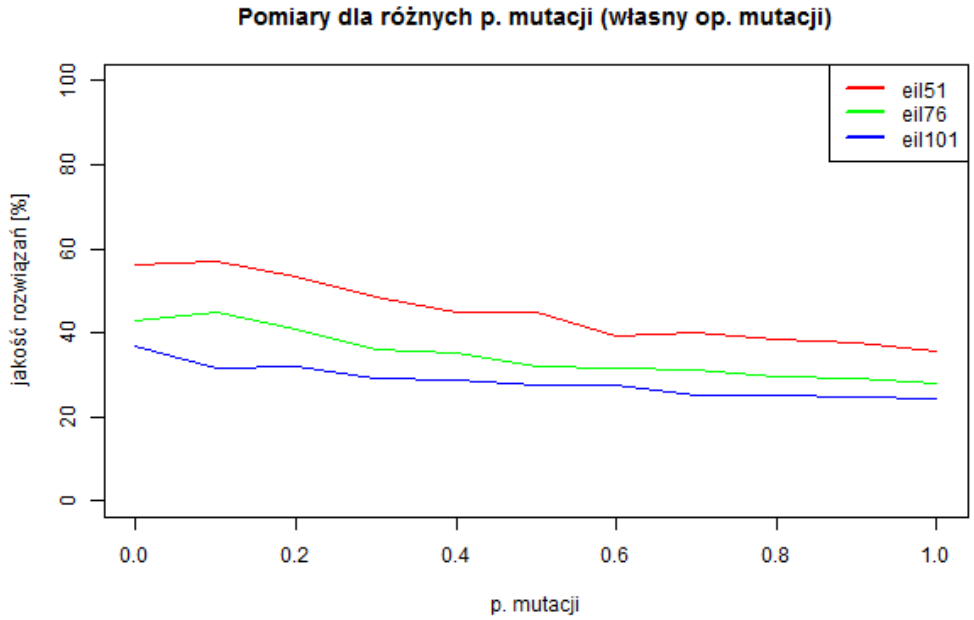
Na ilustracji (rys. 11) przedstawiono wyniki pomiarów dla różnych wartości p. mutacji dla domyślnego operatora.



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

```
# 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) {

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

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

      GAmin <- ga(type = "real-valued",
                  fitness = function(xx) -f(xx),
                  min = c(B[1,]), max = c(B[2,]),
                  optim = TRUE,
                  optimArgs = list (
                    poptim = if (mType == "poptim") value else 0.05,
                    pressel = if (mType == "pressel") value else 0.5))
      solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
      eval <- f(solution[1,])
      sum <- sum + eval
    }
    averages <- c(averages, (sum / nOfRuns))
  }

  if (graphs) {
```

```
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,])
```

```
      sum <- sum + eval
    }
    averages <- c(averages, (sum / nOfRuns))
  }
  temp <- c(temp, averages)
}
result <- matrix(c(temp), nrow = length(GAWithHybridSeries), ncol =
    length(values))

if (graphs) {
  # create standalone graph for each serie
  for (serie in 1:length(GAWithHybridSeries)) {
    legendColors <- rep("darkslateblue", length(GAWithHybridSeries))
    legendColors[serie] = "red"
    # save graph with measurement series to file
    png(file = paste(funcName, mType, serie, ".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")

    lastLine <- NA
    seriesNames <- c()
    for (i in 1:length(GAWithHybridSeries)) {
      seriesNames <- c(seriesNames, GAWithHybridSeries[i])
      if (i != serie)
      {
        lines(values, result[i,], col = "darkslateblue", type = 'l', lwd = 2)
      }
    }
    lines(values, result[serie,], col = "red", type = 'l', lwd = 2)

    legend("topright", seriesNames, lwd=rep(2,length(GAWithHybridSeries)),
        lty=rep(1,length(GAWithHybridSeries)), col = legendColors)
    dev.off()
  }
}

#PSO

customPSOMeasure <- function(values, valueType, xLabel, title) {
  n <- 500 #ilosc czastek
  m.l <- 50 #ilosc przebiegow
  w <- 0.95

  xmin = c(B[1,])
  xmax = c(B[2,])

  vmax <- c(rep(4, length(xmin)))

  g <- function(x) {
    vec <- c()
    for (row in 1:length(x[,1])) {
```

```
      val <- -f(x[row,])
      vec <- c(vec, val)
    }
    vec
  }

  averages <- c()
  for (value in values)
  {
    sum <- 0
    for (i in 1:nOfRuns)
    {
      message(paste("Sekwencja: ", value))
      message(paste("Przebieg: ", i))

      result <- psoptim(FUN=g,
                        n=n,
                        max.loop=m.l,
                        w=w,
                        c1=if (valueType == "c1") value else 0.2,
                        c2=if (valueType == "c2") value else 0.2,
                        xmin=xmin,
                        xmax=xmax,
                        vmax=vmax,
                        seed=NULL,
                        anim=FALSE)

      sum <- sum + f(result$sol)

    }
    averages <- c(averages, (sum / nOfRuns))
  }

  if (graphs) {

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



# perform set of measurements ----
# GA
customGAMeasure(seq(0, 1, 0.1), "mut",
                "p. mutacji", "Znalezione minimum dla różnych p. mutacji")

```

```
customGAMeasure(seq(10, 100, 10), "pop",
                "rozmiar populacji", "Znalezione minimum dla różnych rozmiarów
                    populacji")

# Hybrid
customHybridMeasure(seq(0, 1, 0.01), "poptim",
                    "p. lokalnego searcha", "Znalezione minimum dla różnych
                        poptimów")

customHybridMeasure(seq(0, 1, 0.01), "pressel",
                    "ciśnienie", "Znalezione minimum dla różnych ciśnień")

# Mixed (GA+Hybrid)
customMeasureGAWithHybrid(seq(0, 1, 0.01), "mut",
                          "p.mutacji", "Znalezione minimum dla różnych p. mutacji")


# PSO
customPSOMeasure(seq(0, 1, 0.1), "c1" ,
                 "c1", "Znalezione minimum dla różnych wartości c1")

customPSOMeasure(seq(0, 1, 0.01), "c2" ,
                 "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 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 badania algorytmu genetycznego dla problemu komiwojażera.

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

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

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

numberOfMeasurements <- 15

# instances for testing 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")
```

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

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. W przypadku problemu TSP 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)