Politechnika Wrocławska

Laboratorium

Inteligencja Obliczeniowa i jej Zastosowania

Algorytmy ewolucyjne i hybrydowe

Authors:
Rafał Pieniążek
Jakub Pomykała

Supervisor:
dr hab. inż. Olgierd Unold,
prof. PWr

29 maja 2018

Spis treści

Spis rysunków

1 Wstęp

Celem laboratorium było przeprowadzenie optymalizacji globalnej dla wybranych funkcji z pakietu globalOptTests.

2 Zastosowany algorytm optymalizacji

W laboratorium zastosowano algorytmy genetyczne będące klasą algorytmów ewolucyjnych. Algorytmy ewolucyjne stanowią kierunek sztucznej inteligencji, która wykorzystuje i symuluje ewolucję biologiczną. Wszystkie algorytmy tej klasy symulują podstawowe zachowania w teorii ewolucji biologicznej - procesy selekcji, mutacji i reprodukcji. Zachowanie jednostek zależy od środowiska. Zbiór jednostek nazywa się populacją. Taka populacja ewoluuje zgodnie z regułami selekcji zgodnie z funkcją celu przypisaną do środowiska. Propagowane do kolejnych pokoleń są tylko najbardziej dopasowane osobniki.

2.1 Zastosowane narzędzia implementacji

2.1.1 Język R

R jest językiem programowania i środowiskiem programistycznym, używanym głównie do obliczeń statystycznych i wizualizacji danych, do sztucznej inteligencji a także do ekonomii i innych zagadnień wykorzystujących obliczenia numeryczne. Został stworzony przez Rossa Ihakę i Roberta Gentlemana na Uniwersytecie w Auckland w Nowej Zelandii.

2.1.2 Pakiet GA

Pakiet GA zawiera zestaw funkcji ogólnego przeznaczenia do optymalizacji z wykorzystaniem algorytmów genetycznych. Dostępnych jest kilka operatorów genetycznych, których można łączyć w celu zbadania najlepszych ustawień dla bieżącego zadania.

2.1.3 Pakiet globalOpts

Pakiet zawierający implementację funkcji przydatnych do przeprowadzania testów wydajnościowych algorytmów optymalizacji globalnej.

3 Własne operatory krzyżowania i mutacji

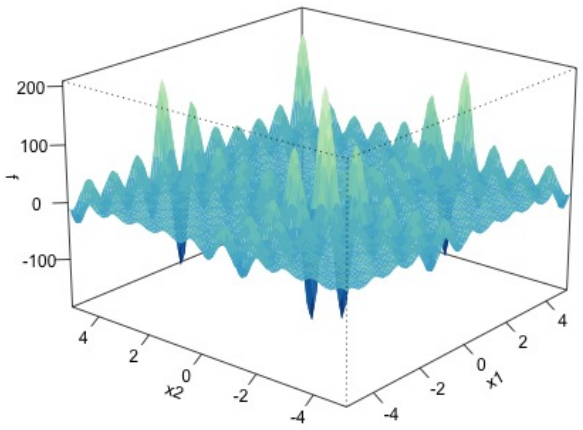
3.1 Funkcja wielomodalna - Funkcja Shuberta

3.1.1 Wzór analityczny

$$f(\mathbf{x}) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right)$$

Rysunek 1: Wzór analityczny funkcji Schuberta

3.1.2 Wykres w ustalonym przedziale zmiennych

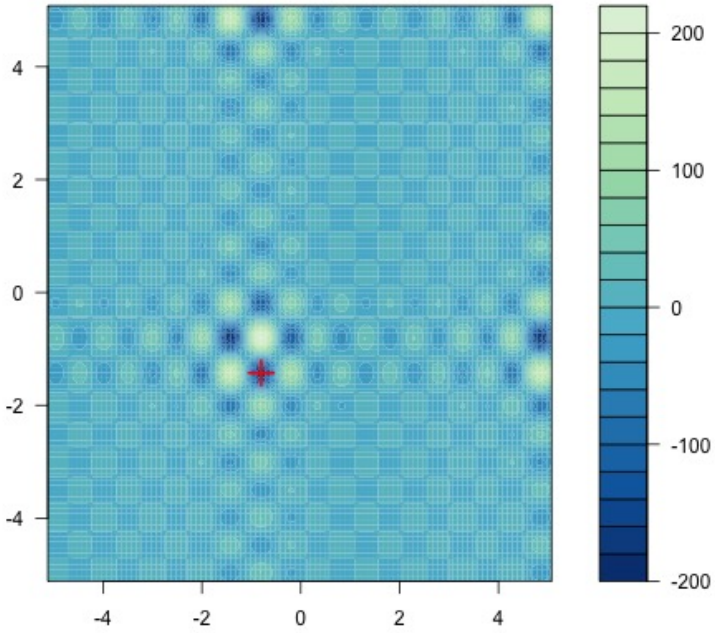


Rysunek 2: Wykres funkcji Schuberta

3.1.3 Ekstremum globalne

$$f(\mathbf{x}^*) = -186.7309$$

Rysunek 3: Minimum globalne dla funkcji Schuberta



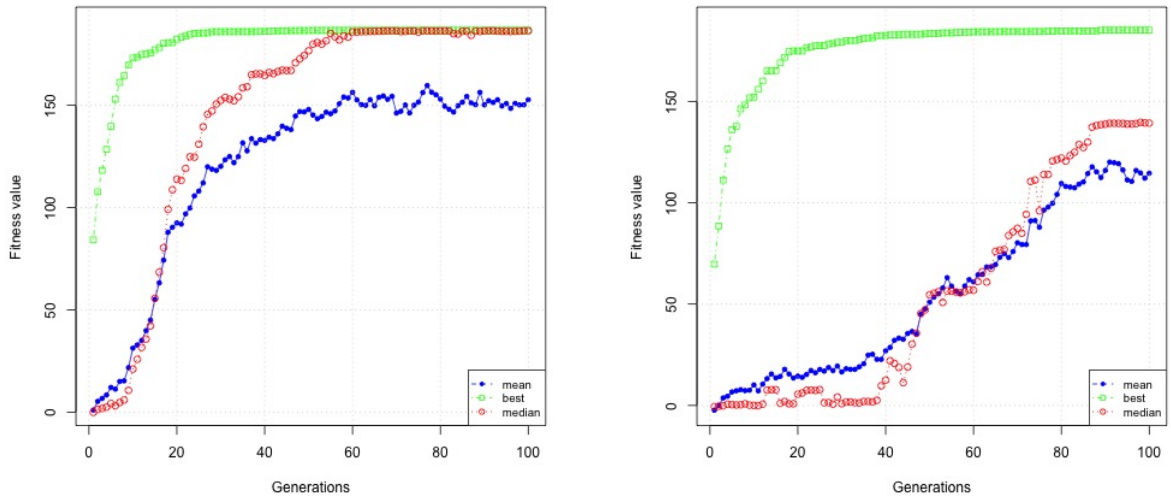
Rysunek 4: Minimum globalne dla funkcji Schuberta

3.2 Zmiana funkcji mutowania

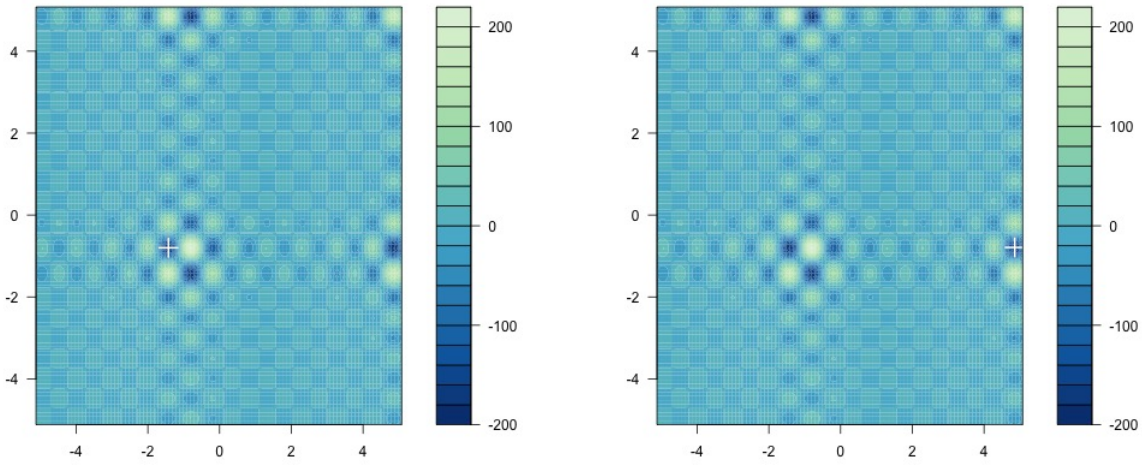
Poniżej przedstawiono własną propozycję implementacji funkcji mutowania.

```
customMutation <- function(object, parent)
    {
      mod <- parent %% 2
      if(mod == 0){
        return (parent * 2)
      } else {
        return (parent / 2) + 1
      }
    }
```

Na poniższych wykresach przedstawiono zestawienie rezultatów działania w przypadku domyślnej i własnej, zaimplementowanej funkcji mutacji. Wyniki dla funkcji domyślnej znajdują się po lewej stronie.



Rysunek 5: Porównanie domyślnej funkcji (lewa strona) mutowania z własną (prawa strona)



Rysunek 6: Porównanie domyślnej funkcji mutowania (lewa strona) z własną (prawa strona) - znalezione ekstrema

Zmiana funkcji mutującej nie wpłynęła znacząco na wyniki końcowe. Mimo, iż w początkowej fazie wynik funkcji zbiegał do ekstremum wolniej niż w przypadku funkcji domyślnej, po 100 pokoleniach wynik jest na podobnym poziomie w obu przypadkach. Finalnie jednak algorytm z własną funkcją mutującą nie znalazł ekstremum globalnego. Widoczne jest to na wykresie temperaturowym, na którym zaznaczono rozwiązanie znalezione przez algorytm.

W przypadku funkcji z własną funkcją mutowania można zauważyć spadek średniej i mediany wyników. Jest to spowodowane tym że dane zostają wyznaczane stochastycznie przez pakiet GA, a ich modyfikacja przez pomnożenie lub podzielenie przez stałą liczbę nie może poprawić ani pogorszyć wyniku w tym przypadku. Jakość znalezionych rozwiązań będzie mocno skorelowana z pierwotnie wyznaczonymi wartościami.

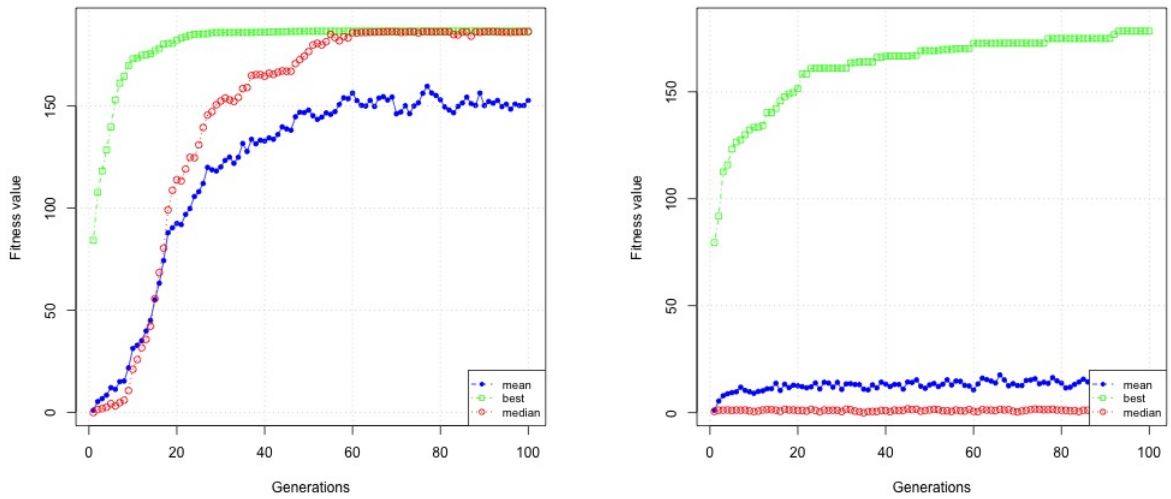
3.3 Zmiana funkcji krzyżowania

Poniżej przedstawiono własną propozycję implementacji funkcji krzyżowania.

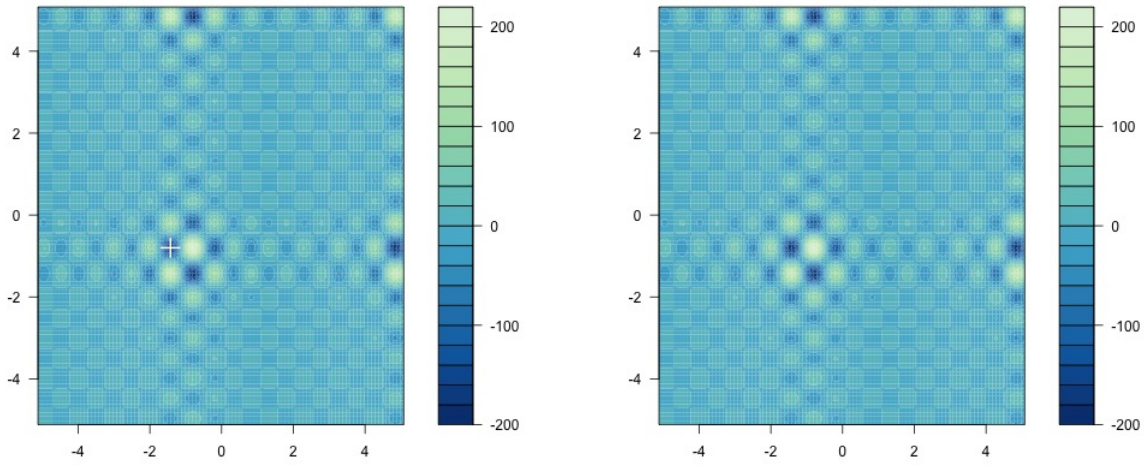
```
customCrossover <- function(object, parents)
    {
      parent_1 <- parents[[1]]
      parent_2 <- parents[[2]]
      wektor_1 <- c(parent_1, parent_2)
      fitness = testFunctionWrapper(parent_1, parent_2)

      tmp_parent_1 <- parents[[1]] + runif(1, -1, 1)
      tmp_parent_2 <- parents[[2]] + runif(1, 1, -1)
      tmp_wektor_1 <- c(parent_1, parent_2)
      tmp_fitness = testFunctionWrapper(parent_1, parent_2)

      if(tmp_fitness > fitness){
        return (list(children=matrix(tmp_wektor_1), fitness=tmp_fitness))
      }
      return (list(children=matrix(wektor_1), fitness=fitness))
```

}

Na poniższych wykresach przedstawiono porównanie rezultatu działania algorytmu w przypadku zmiany funkcji krzyżowania z domyślnej na własną implementację.



Rysunek 7: Porównanie domyślnej funkcji krzyżowania (lewa strona) z własną (prawa strona)



Rysunek 8: Porównanie domyślnej funkcji krzyżowania (lewa strona) z własną (prawa strona) - znalezione ekstrema

W przypadku własnej funkcji krzyżującej algorytm nie znalazł ekstremum globalnego. Wartości średniej i mediany spadły znaczącą w stosunku do domyślnej implementacji. Mediana jest praktycznie równa zero. Algorytm prawdopodobnie znalazł minimum lokalne i nie mógł odnaleźć

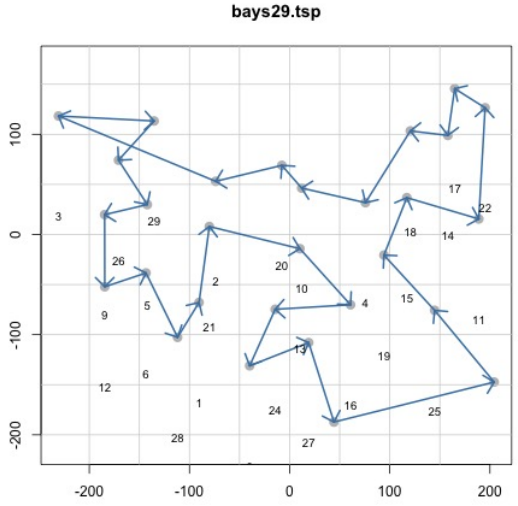
lepszych rozwiązań, mimo wybierania lepszej wartości *fitness*, ponieważ do każdego rodzica jest dodawana wartość losowa z przedziału [-1, 1]. Prawdopodobnie gdyby ten przedział był większy algorytm mógłby wylosować lepszego rodzica, który nie znajduje się w lokalnym minimum.

4 Problem komiwojażera

4.1 Opis problemu

Problem komiwojażera jest jednym z najsłynniejszych problemów informatyki i badań operacyjnych. Często nazywa się skrótem TSP, skrót nazwy angielskiej nazwy "podróżujący sprzedawca". Można go sformułować w następujący sposób: "Biorąc pod uwagę n miast wraz z odległością między każdą parą tych miast, znajdź najkrótszą trasę, która przebiega przez każde miasto dokładnie raz."

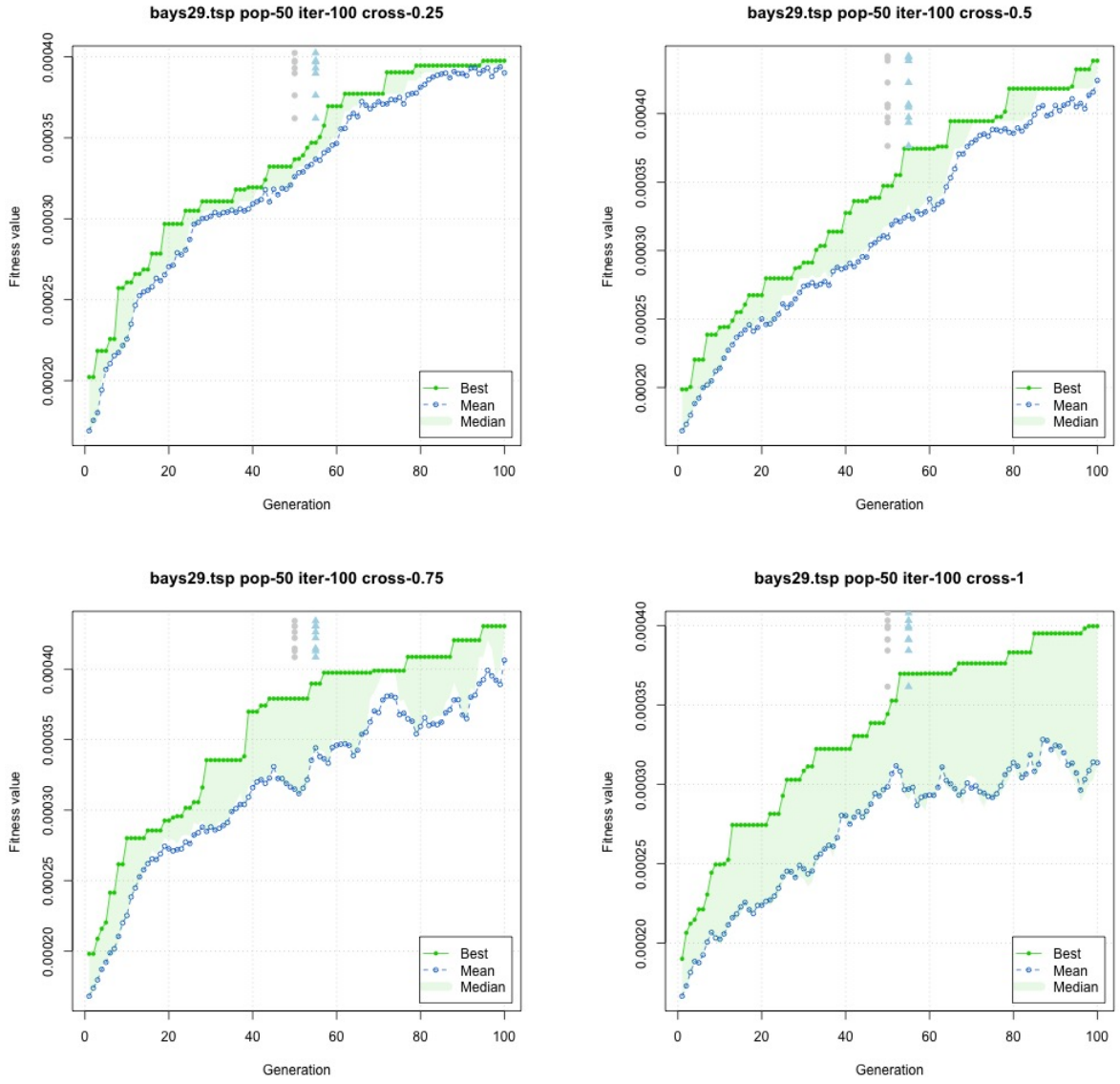
4.2 Instancja 1 - *bays29*



Rysunek 9: Rozwiązanie dla domyślnych parametrów

4.2.1 Zmiana parametru krzyżowania

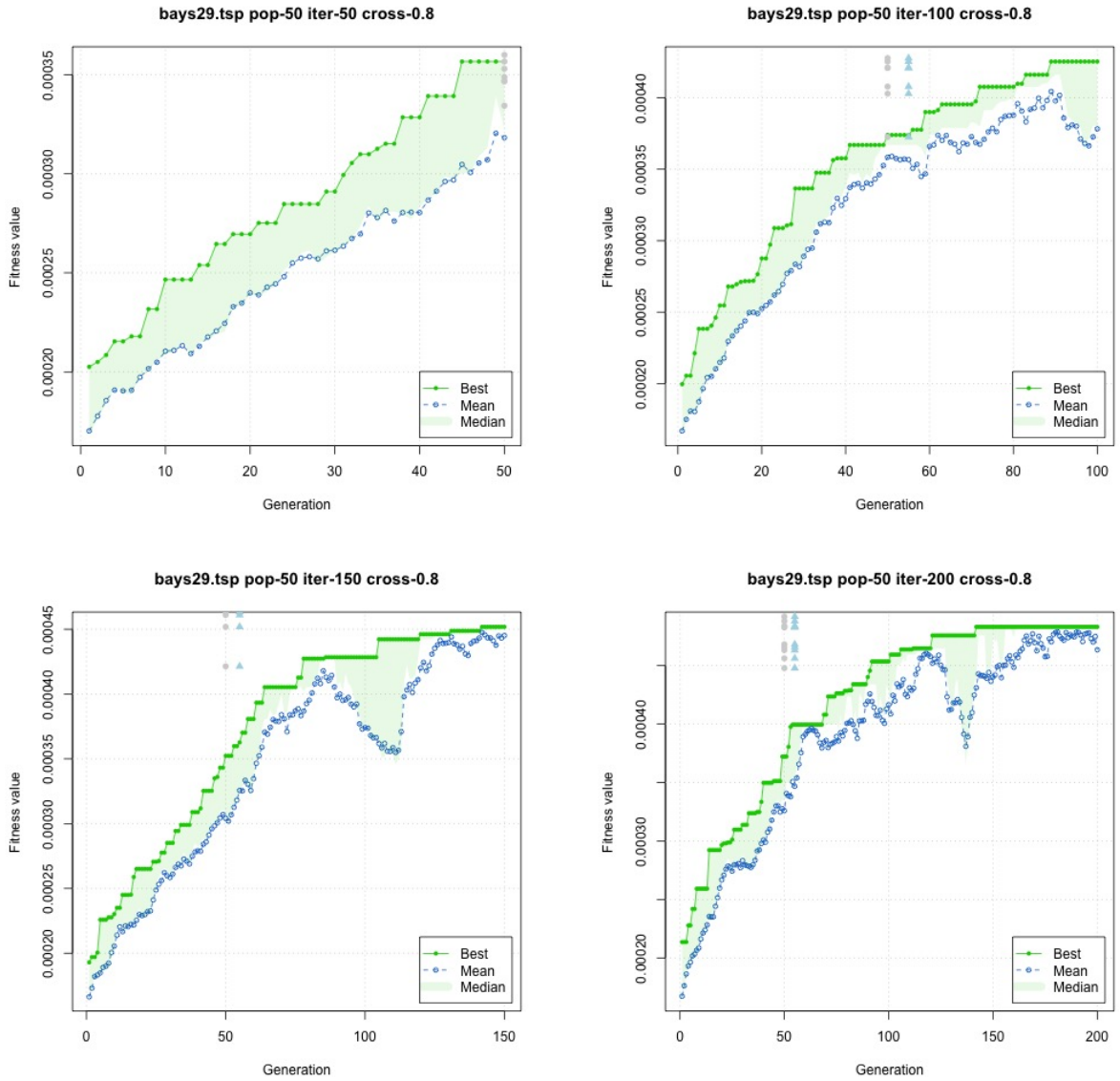
Zwiększanie parametru krzyżowania zmniejsza wartość średniej i mediany, oznacza to, że średnie wyniki w populacji są coraz gorsze. Algorytm znalazł najlepsze rozwiązanie dla wartości parametru 0.5.



Rysunek 10: Porównanie wyników podczas zmiany parametru krzyżowania

4.2.2 Zmiana parametru liczby pokoleń

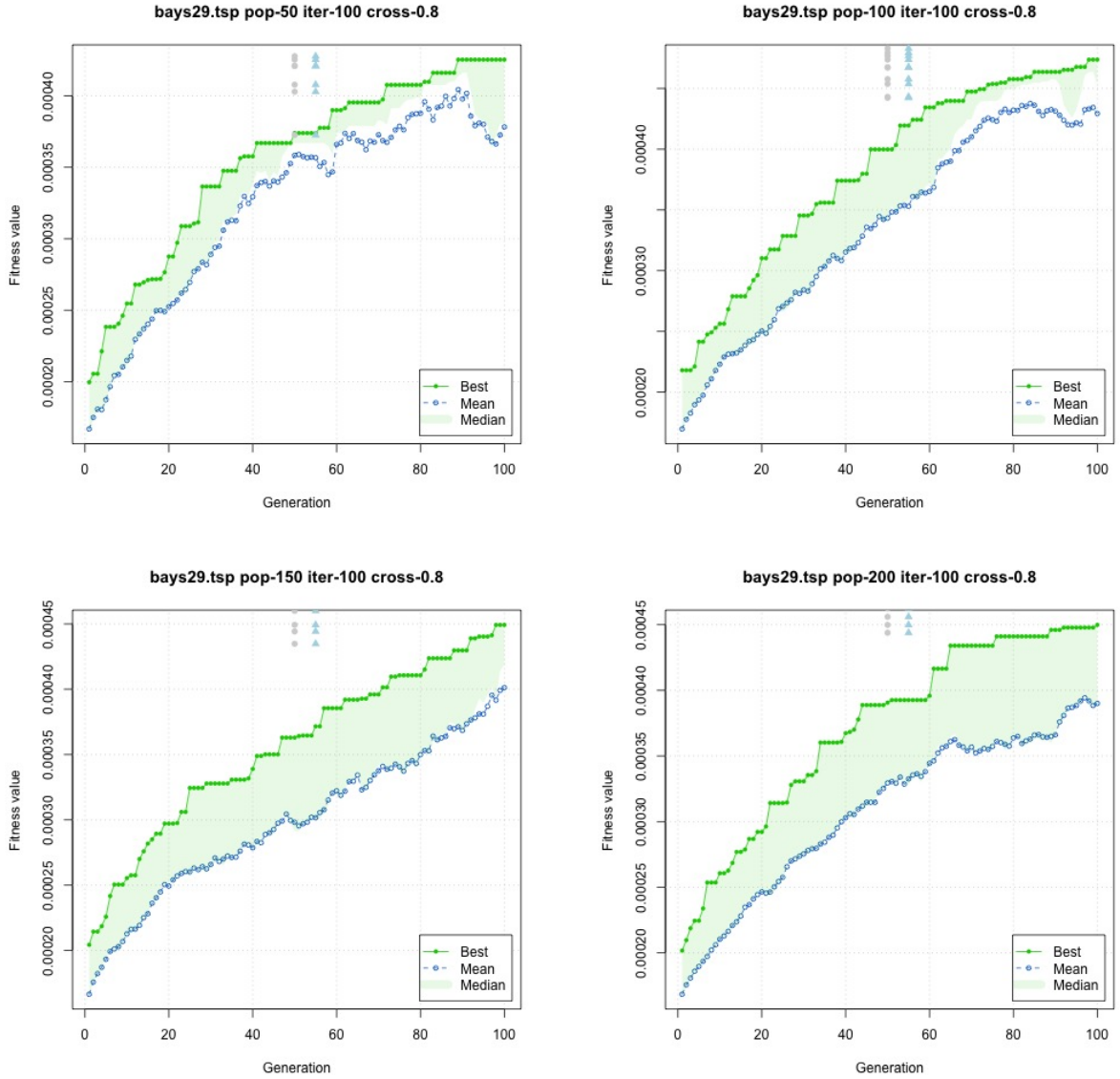
Zwiększanie parametru liczby pokoleń zwiększa wartość znajdowanego rozwiązania. Poziom wartości średniej i mediany są bliskie najlepszemu rozwiązaniu.



Rysunek 11: Porównanie wyników podczas zmiany parametru liczby pokoleń

4.2.3 Zmiana parametru rozmiaru populacji

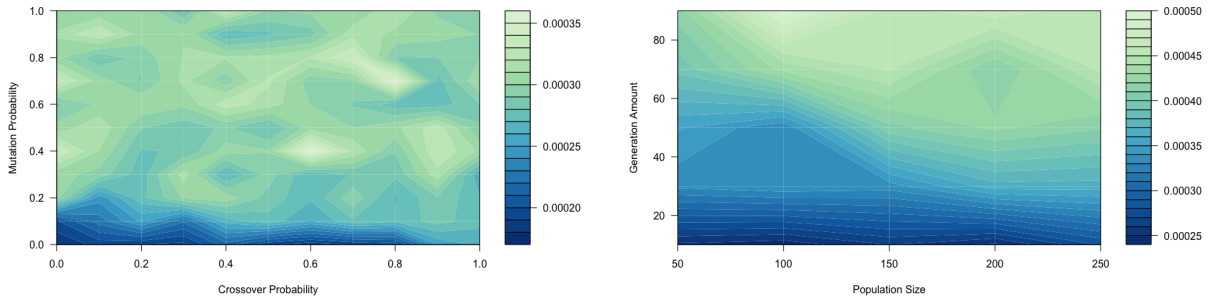
Wraz ze wzrostem ilości osobników w pokoleniu algorytm znajduje coraz lepsze rozwiązania. Wartość średniej i mediany maleje wraz ze wzrostem rozmiaru populacji.



Rysunek 12: Porównanie wyników podczas zmiany parametru rozmiaru populacji

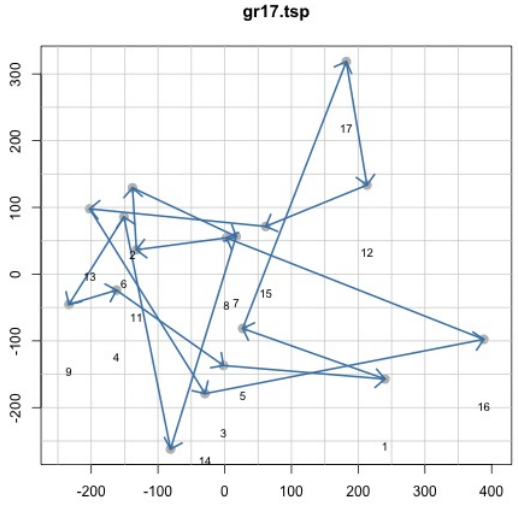
4.2.4 Jednoczesna zmiana krzyżowania i mutacji oraz rozmiaru populacji i liczby pokoleń

Algorytm uzyskał najlepszy wynik gdy populacja wynosiła 100, a liczba pokoleń 80. Analiza wykresu temperaturowego jednoczesnej zmiany mutowania i krzyżowania nie pozwala wysunąć jednoznacznych wniosków. Można wydzielić dwa obszary parametrów dla których algorytm znalazł lepsze rozwiązanie: krzyżowanie 0.6 i mutacja 0.4 oraz krzyżowanie 0.8 i mutacja 0.75



Rysunek 13: Porównanie wyników podczas jednoczesnej zmiany parametrów

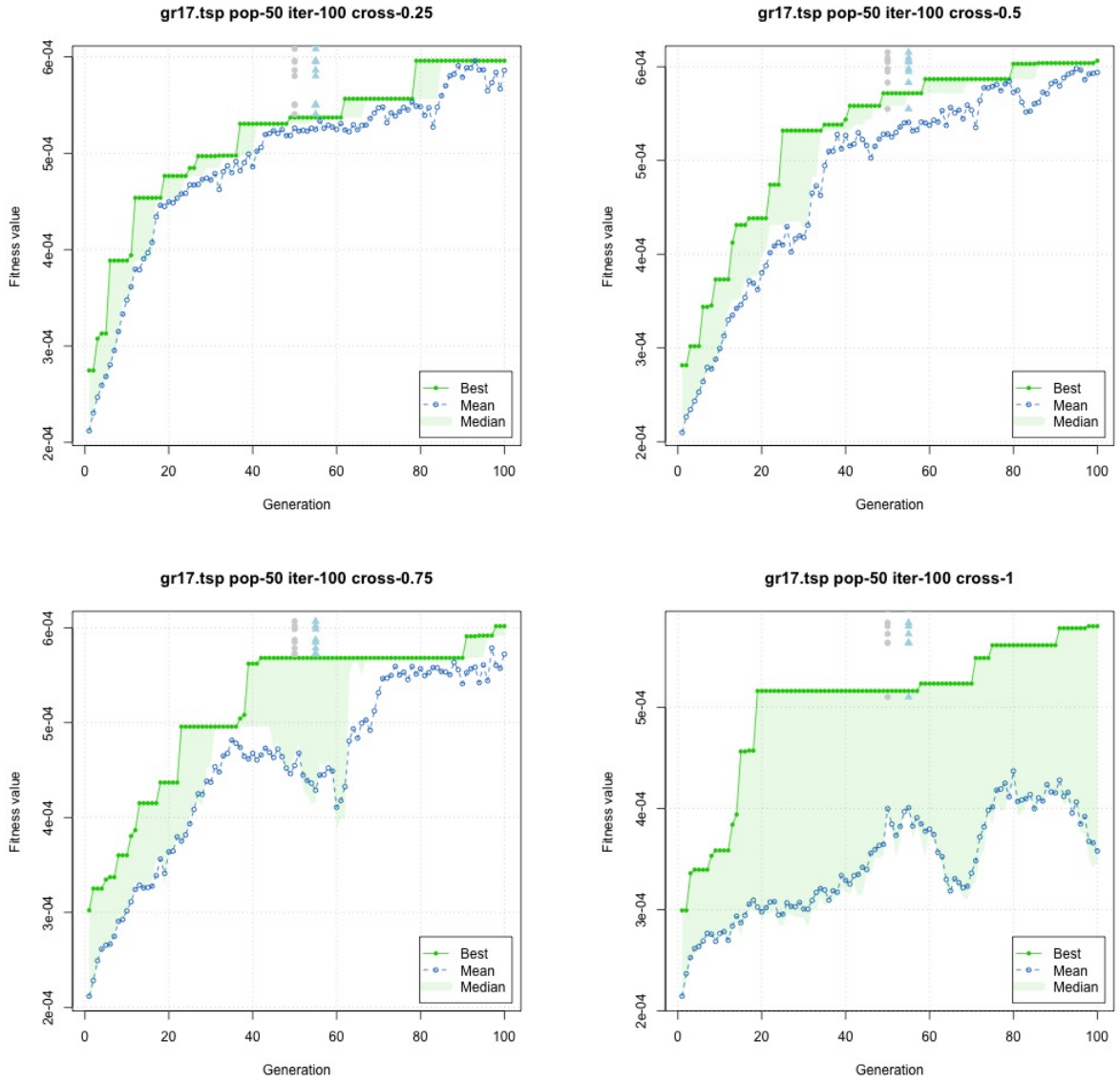
4.3 Instancja 2 - *gr17*



Rysunek 14: Rozwiązanie dla domyślnych parametrów

4.3.1 Zmiana parametru krzyżowania

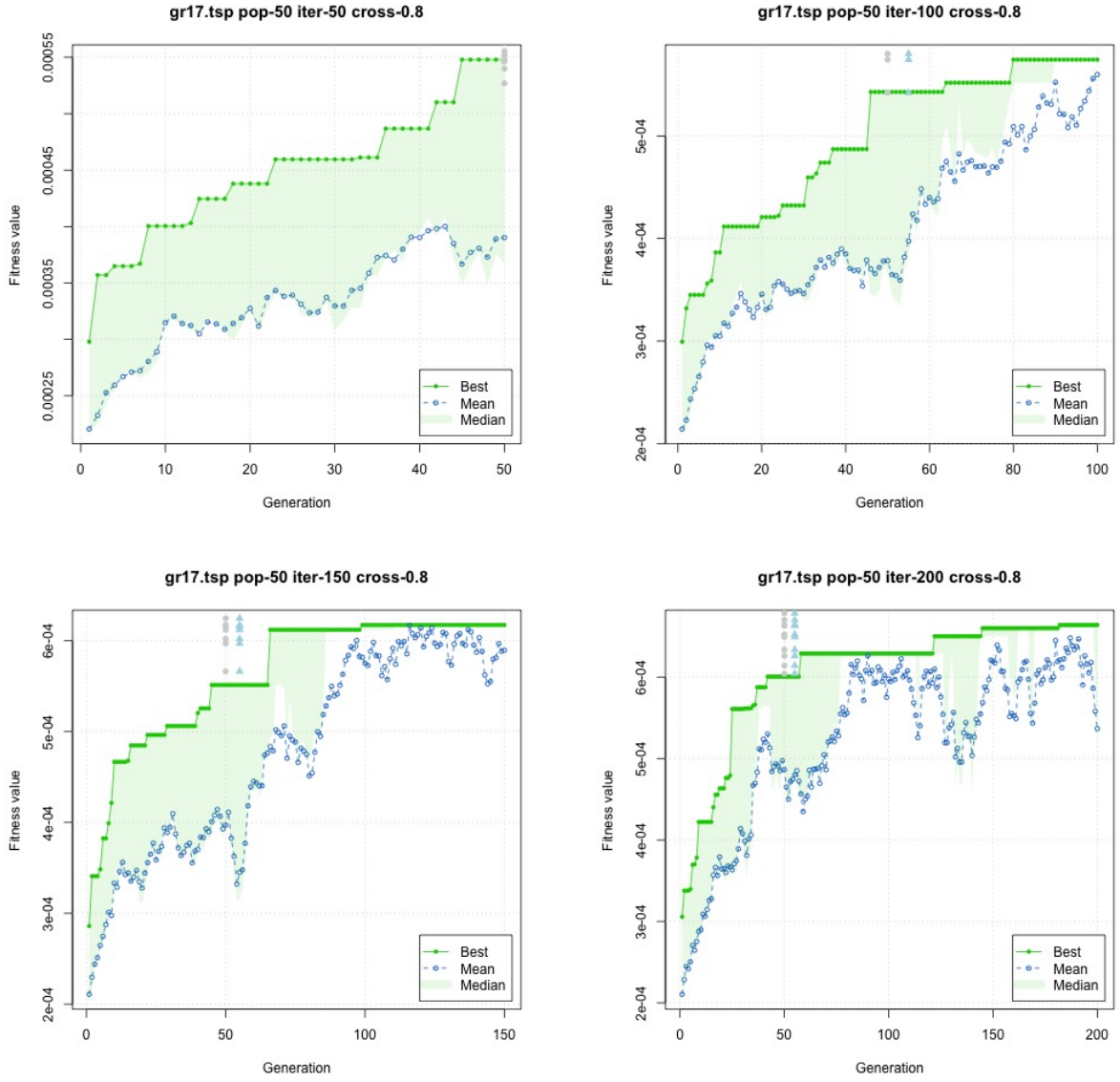
Średnie wartości są zbliżone do najlepszego rozwiązania dla najmniejszych wartości parametru krzyżowania. Wzrost tego parametru skutkuje zmniejszeniem średniej wyników w każdej populacji.



Rysunek 15: Porównanie wyników podczas zmiany parametru krzyżowania

4.3.2 Zmiana parametru liczby pokoleń

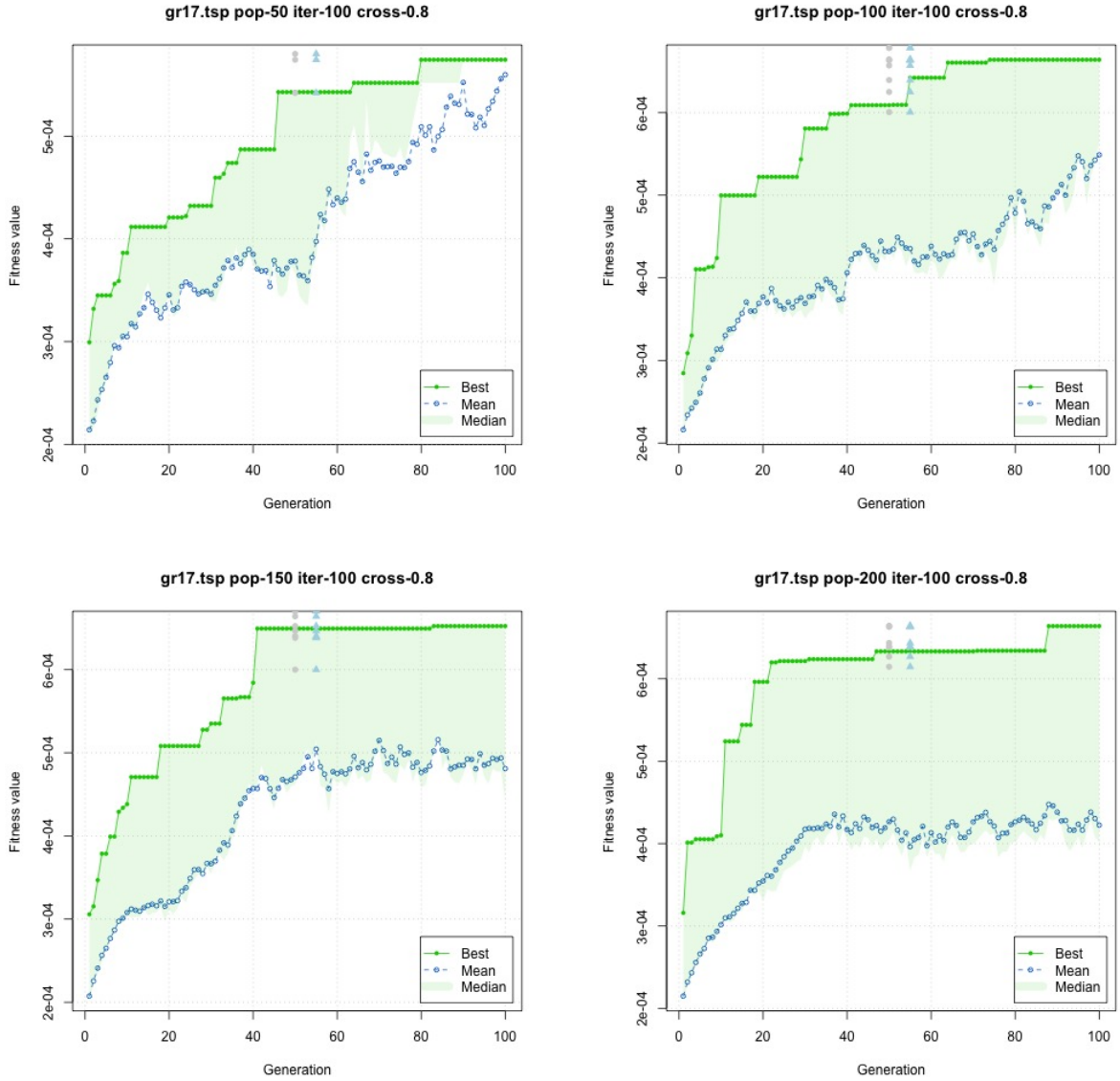
Wzrost parametru liczby pokoleń jest jednoznaczny ze wzrostem wartości najlepszego rozwiązania. Zmniejsza się również wartość średnia i mediana.



Rysunek 16: Porównanie wyników podczas zmiany parametru liczby pokoleń

4.3.3 Zmiana parametru rozmiaru populacji

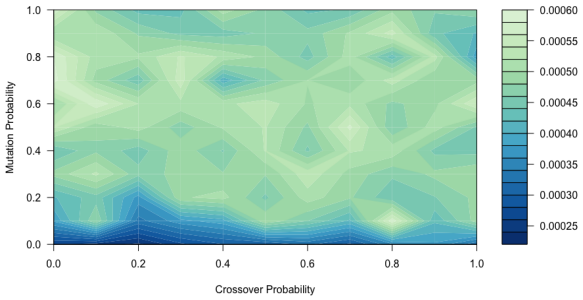
Wartość znajdowanego rozwiązania jest na stałym poziomie niezależnie od ilości osobników. Zmniejsza się jednak średnia wartość dla każdego pokolenia.



Rysunek 17: Porównanie wyników podczas zmiany parametru rozmiaru populacji

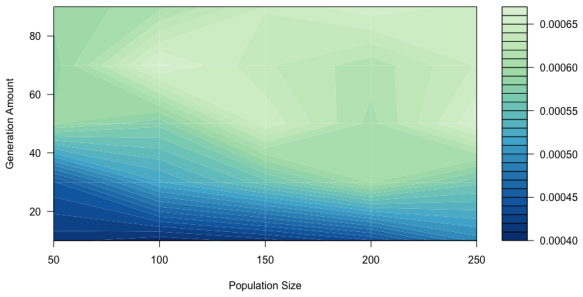
4.3.4 Jednoczesna zmiana krzyżowania i mutacji oraz rozmiaru populacji i liczby pokoleń

Dla tej instancji nie znaleziono korelacji między jednoczesną zmianą wartości krzyżowania i mutacji a polepszeniem uzyskiwanego wyniku. Najlepszy wynik uzyskano dla wartości krzyżowania 0.8 i mutacji 0.1.



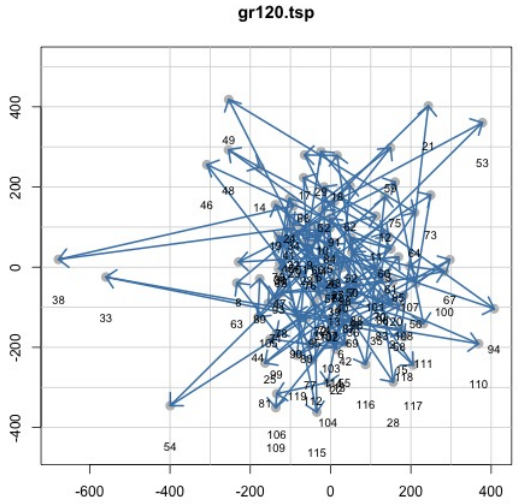
Rysunek 18: Porównanie wyników podczas jednoczesnej zmiany parametrów

Dla jednoczesnej zmiany rozmiaru populacji i liczby iteracji najlepszy wynik uzyskano dla 100 osobników i 75 pokoleń.



Rysunek 19: Porównanie wyników podczas jednoczesnej zmiany parametrów

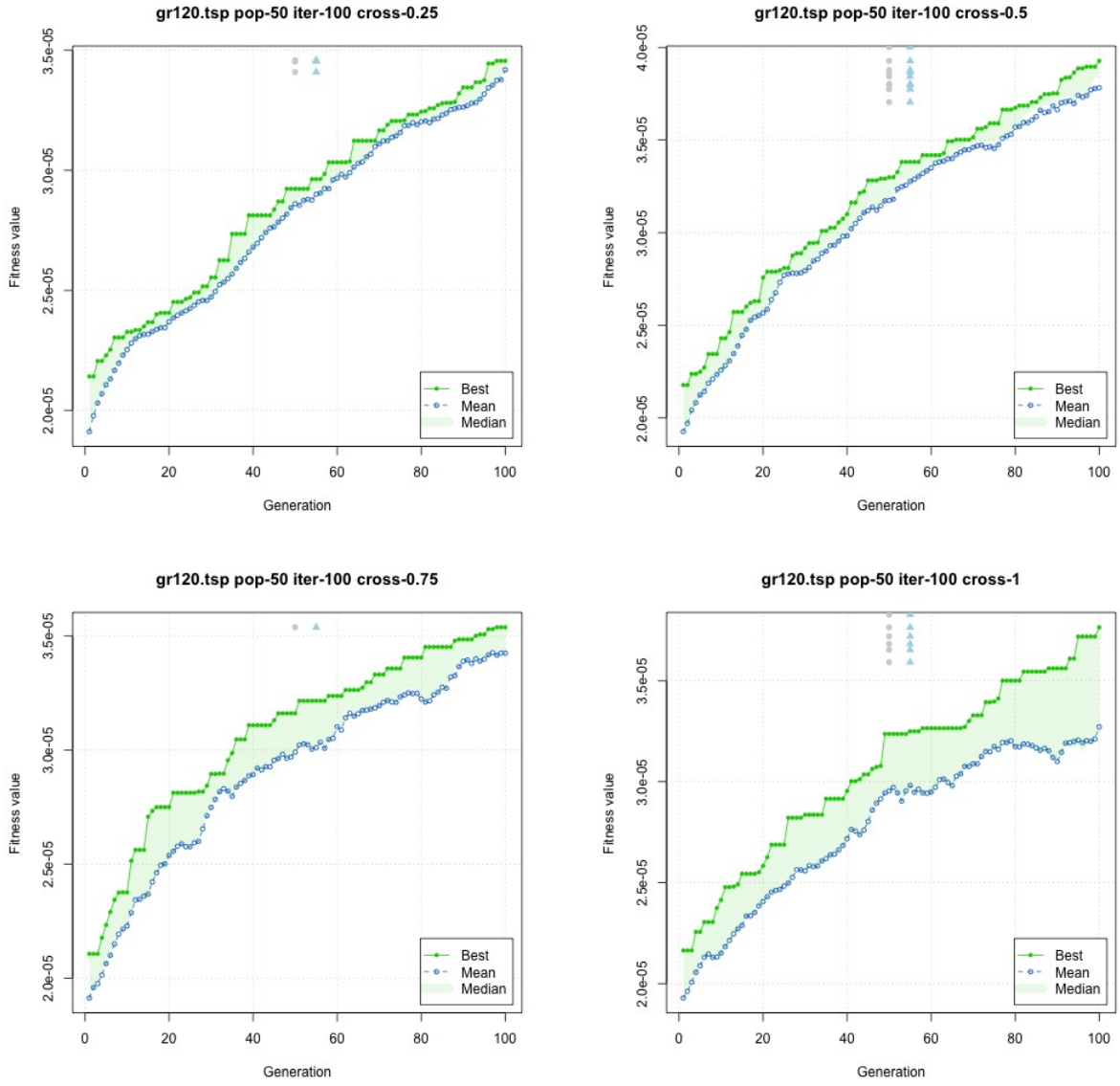
4.4 Instancja 3 - *gr120*



Rysunek 20: Rozwiązanie dla domyślnych parametrów

4.4.1 Zmiana parametru krzyżowania

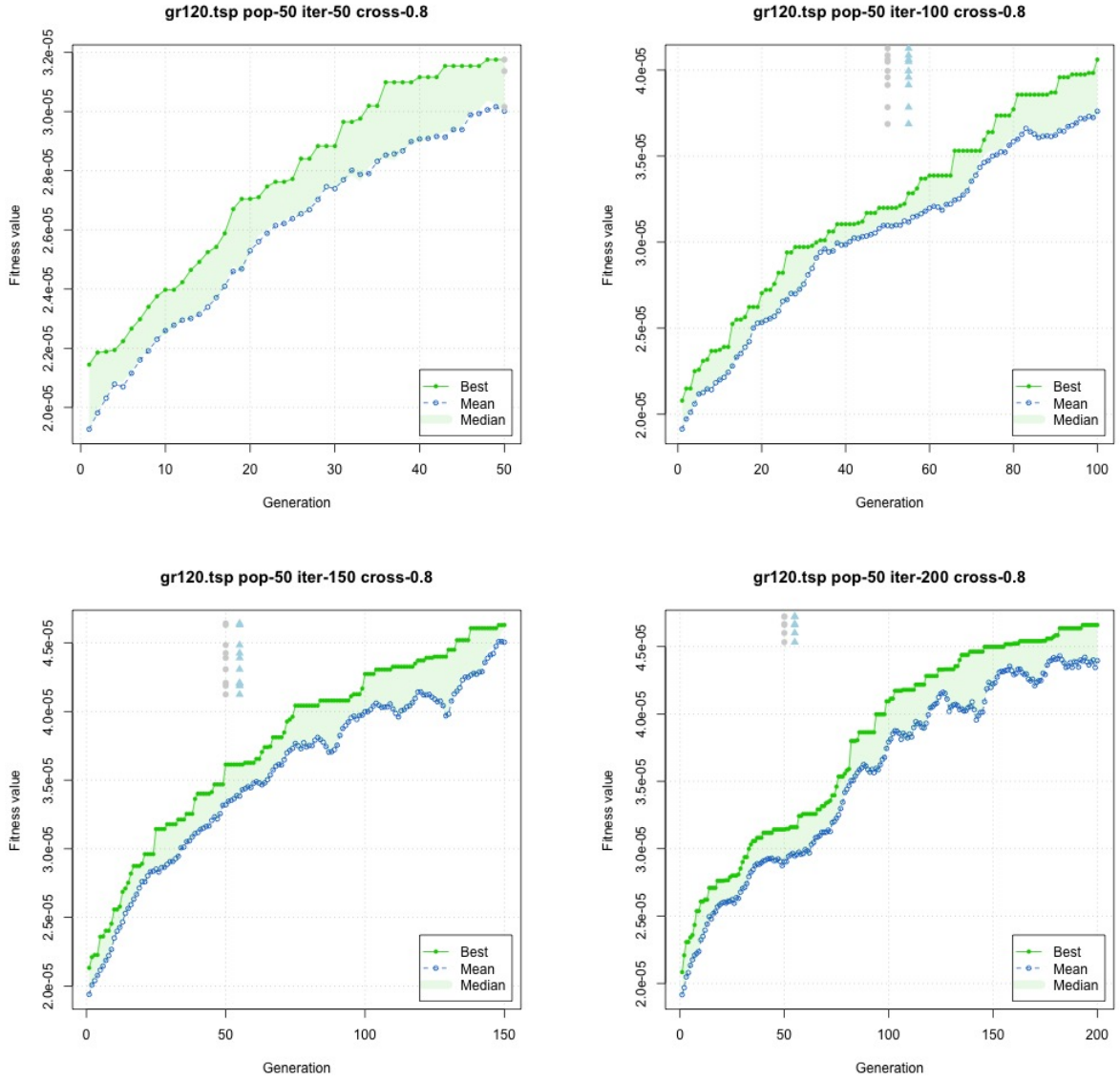
Średnie wartości są zbliżone do najlepszego rozwiązania dla najmniejszych wartości parametru krzyżowania. Wzrost tego parametru skutkuje zmniejszeniem średniej wyników w każdej populacji.



Rysunek 21: Porównanie wyników podczas zmiany parametru krzyżowania

4.4.2 Zmiana parametru liczby pokoleń

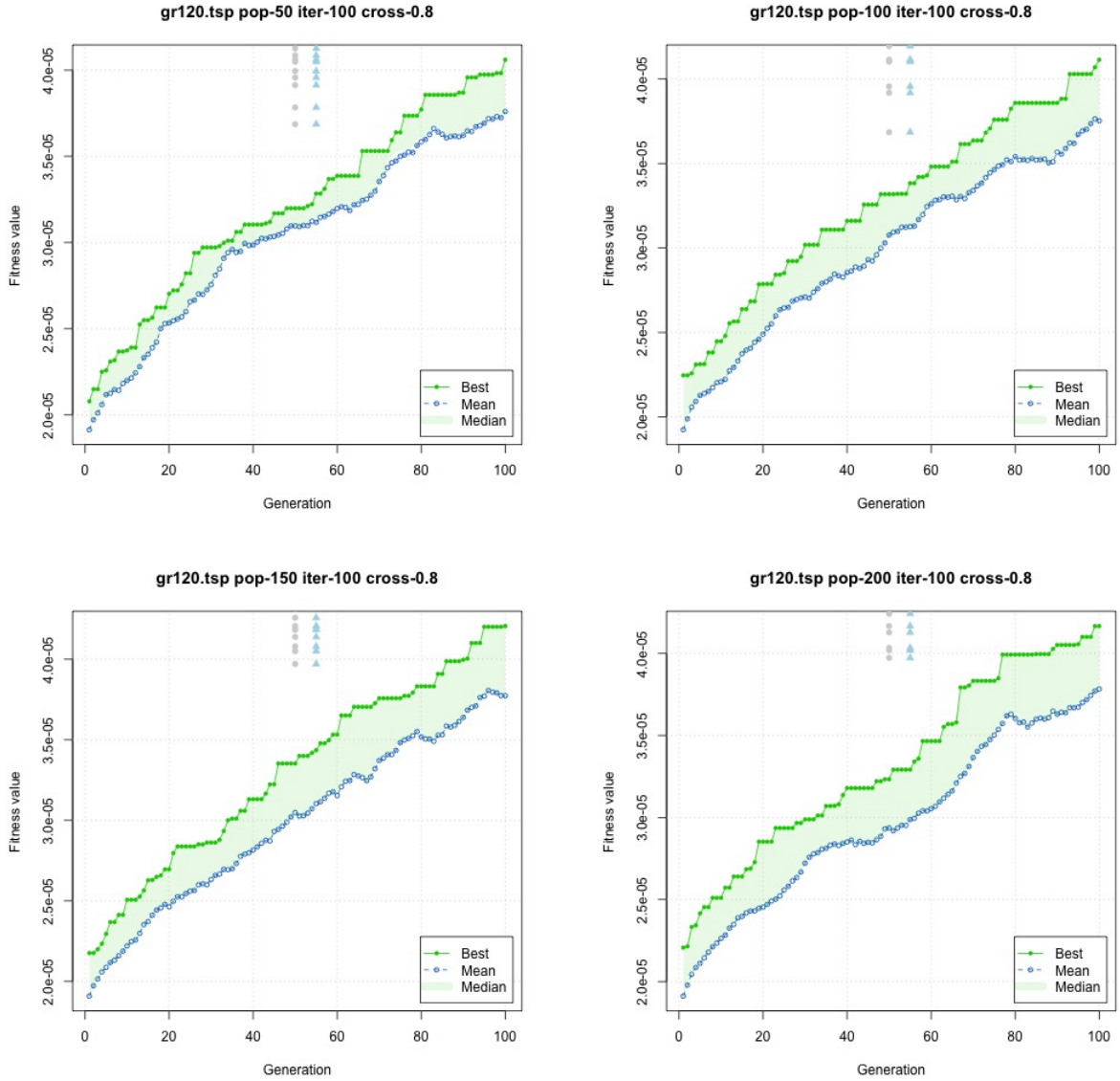
Wzrost parametru liczby pokoleń jest jednoznaczny ze wzrostem wartości najlepszego rozwiązania. Zmniejsza się również wartość średnia i mediana.



Rysunek 22: Porównanie wyników podczas zmiany parametru liczby pokoleń

4.4.3 Zmiana parametru rozmiaru populacji

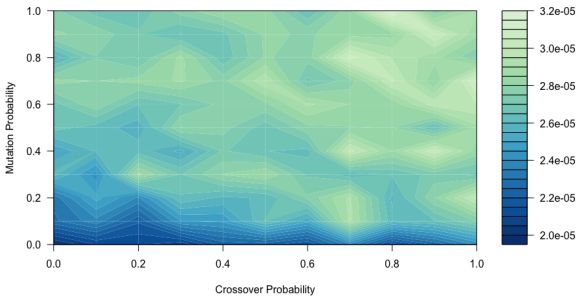
Wartość znajdowanego rozwiązania jest na stałym poziomie niezależnie od ilości osobników. Zmniejsza się jednak średnia wartość dla każdego pokolenia.



Rysunek 23: Porównanie wyników podczas zmiany parametru rozmiaru populacji

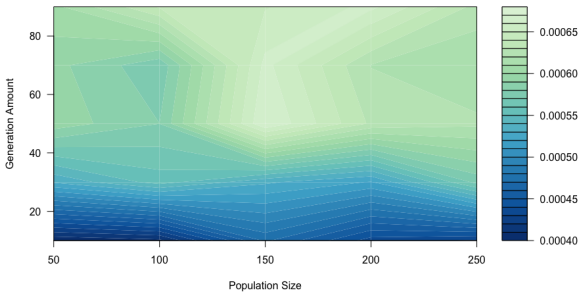
4.4.4 Jednoczesna zmiana krzyżowania i mutacji oraz rozmiaru populacji i liczby pokoleń

Dla tej instancji nie znaleziono korelacji między jednoczesną zmianą wartości krzyżowania i mutacji a polepszeniem uzyskiwanego wyniku.



Rysunek 24: Porównanie wyników podczas jednoczesnej zmiany parametrów

Dla jednoczesnej zmiany rozmiaru populacji i liczby iteracji najlepszy wynik uzyskano dla 150 osobników i 50 pokoleń.



Rysunek 25: Porównanie wyników podczas jednoczesnej zmiany parametrów

4.5 Wnioski

4.5.1 Zmiana parametru populacji

Zwiększenie rozmiaru populacji daje większą szansę znalezienia optymalnego rozwiązania po przez zagęszczenie przeszukiwań na danej powierzchni.

4.5.2 Zmiana parametru krzyżowania

Zwiększenie parametru krzyżowania powoduje zwiększenie ilości osobników z gorszymi wynikami. Jest to powód dla którego zwiększyła się wartości wariancji dla wartości średniej.

4.5.3 Zmiana wartości liczby pokoleń

Zwiększenie parametru liczby pokoleń wpywają na jakość znajdowanych rozwiązań ponieważ algorytm ma więcej możliwści na znalezienie bardziej optymalnego rozwiązania.

4.5.4 Jednoczesna zmiana wartości krzyżowania i mutacji

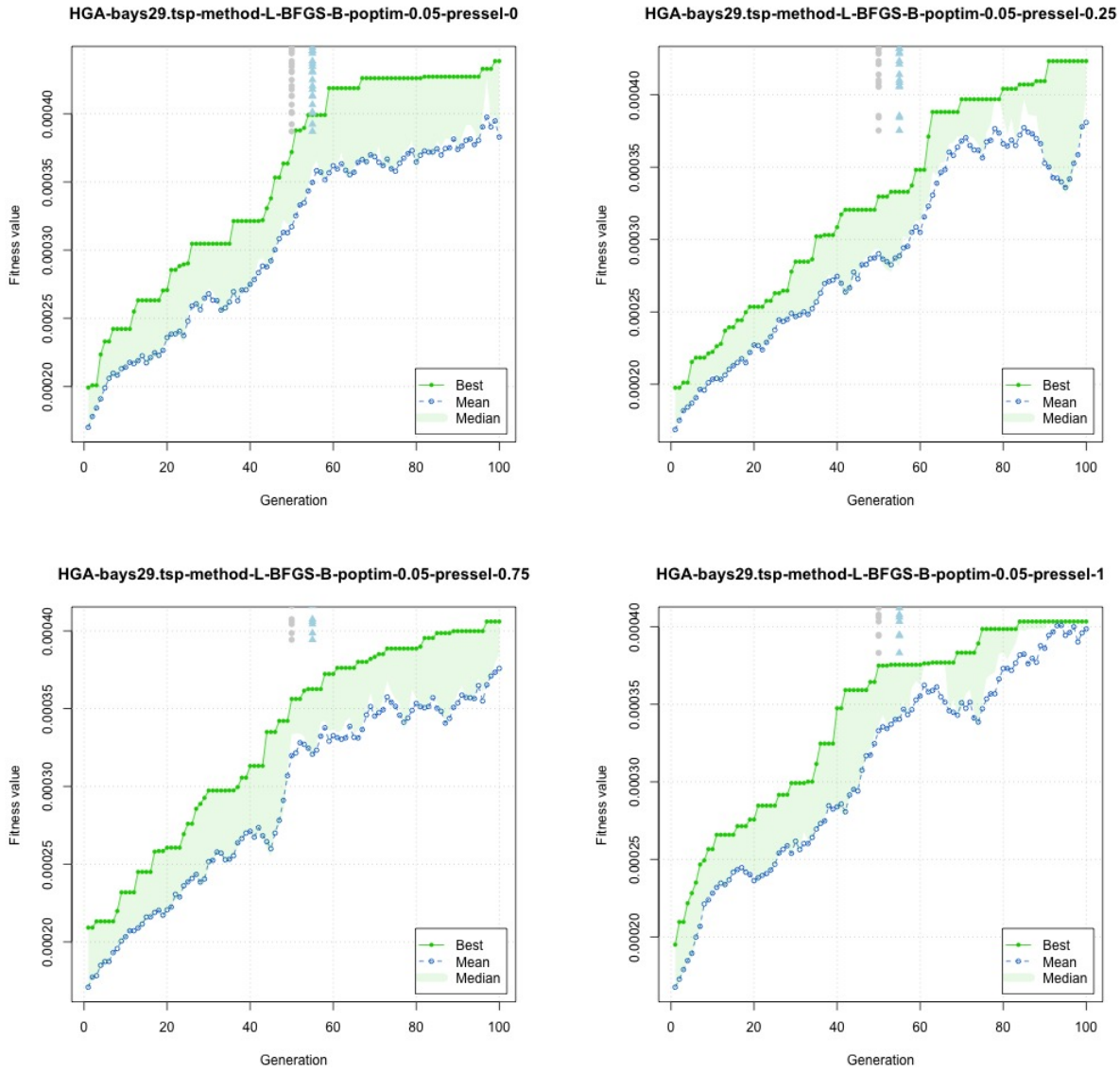
Temperaturowy wykres funkcji krzyżowania i mutacji ma charakter multimodalny. Zerowa wartość parametru mutacji powoduje generowanie wyników nieoptymalnych, nie zależnie od parametru prawdopodobieństwa krzyżowania. Z pomocą wygenerowanych wykresów można znaleźć miejsca w których parametry krzyżowania i mutacji są optymalne dla danej instancji.

4.5.5 Jednoczesna zmiana rozmiaru populacji i liczby pokoleń

Wyniki pokazują że w każdym przypadku liczba iteracji miała większy wpływ na poprawę wyników niż liczba osobników populacji.

5 Hybrydowy algorytm genetyczny

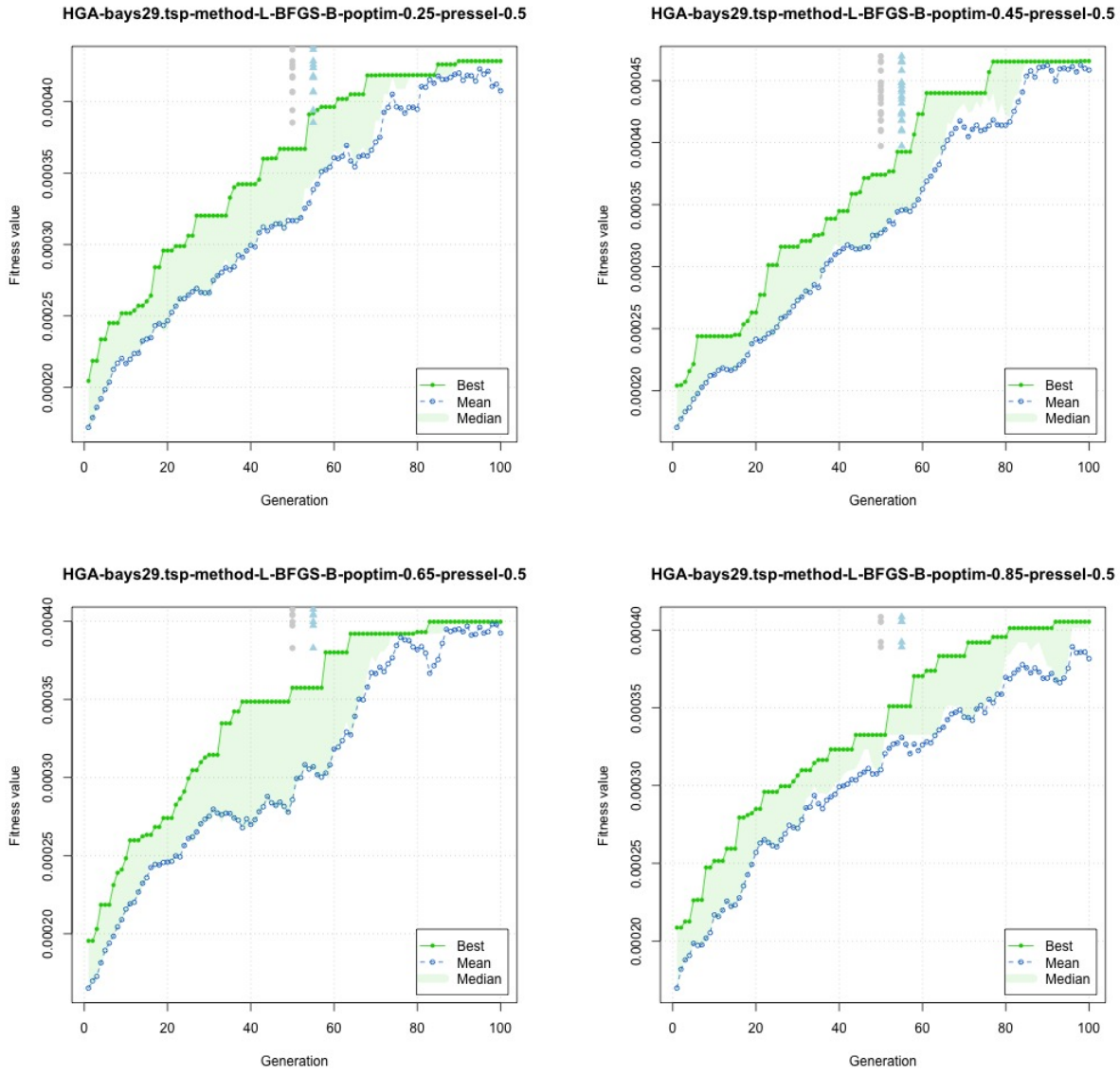
5.1 Zmiana parametru *pressel*



Rysunek 26: Porównanie wyników podczas zmiany parametru nacisku

Mniejsze wartości parametru *pressel* mają tendencję do przypisywania równych prawdopodobieństw wszystkim rozwiązaniom, a większe wartości mają tendencję do przypisywania większych wartości tym rozwiązaniom o lepszej wartości dopasowania. Gdy parametr *pressel* ma wartość 0, to wszystkim minimom lokalnym przyporządkowano to samo prawdopodobieństwo. Większe prawdopodobieństwa są przypisywane do większych wartości *fitness*, gdy wzrasta wartość parametru *pressel*.

5.2 Zmiana parametru *poptim*



Rysunek 27: Porównanie wyników podczas zmiany parametru prawdopodobieństwa

W implementacji dostępnej w pakiecie GA, wyszukiwanie lokalne jest stosowane stochastycznie podczas iteracji GA z prawdopodobieństwem *poptim* od 0 do 1. Tendencje wykresów zmiany wykresów parametru *pressel* są analogiczne do zmian parametru *poptim*. Większe wartości parametru *poptim* algorytm szybciej znajduje rozwiązania optymalne.

6 Literatura

1. Artur Suchwałko, "Wprowadzenie do R dla programistow innych jezykow", https://cran.r-project.org/doc/contrib/R-dla-programistow-innych-jezykow.pdf, 2014-02-23

2. Luca Scrucca, "On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution", https://arxiv.org/pdf/1605.01931.pdf, 2016-05-09

3. dr inż. Julian Sienkiewicz, "Pakiet R w analizie układów złożonych", http://www.if.pw.edu.pl/~julas/CSAR/csar11.html, 2017

4. W. N. Venables, D. M. Smith, R Core Team, "An Introduction to R", https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf, 2018-04-23

5. Luca Scrucca, "Package 'GA'", ftp://cran.r-project.org/pub/R/web/packages/GA/GA.pdf, 2016-09-29

6. Katharine Mullen, "Package 'globalOptTests'", https://cran.r-project.org/web/packages/globalOptTests/globalOptTests.pdf, 2015-02-15

7. Abdal-Rahman Hedar, "Global Optimization Test Problems", http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm, dostęp online: 2018-05-04

7 Kod źródłowy

```
#https://cran.r-project.org/web/packages/GA/vignettes/GA.html
#install.packages("GA");# do instalacji biblioteki GA
#install.packages("globalOptTests")
library(GA)
library(globalOptTests)

#miejsce zapisu wykresow
path = '/Users/evelan/Desktop/ga.nosync/'

#uzyte funkcji
fnNames = c("Schubert")

# liczba przebiegow
testInstances = 20

#domyslne parametry
defaultPopSize = 50
defaultCrossover = 0.8
defaultMutation = 0.1
defaultElitePopulation = 0.05
defaultIterationSize = 100

populationSizes = seq(50, 250, by = 50)
iterSizes = seq(50, 250, by = 50)
crossoverSizes = seq(0, 1.0, by = 0.25)
mutationSizes = seq(0, 1.0, by = 0.25)
elitePopulationSizes = seq(0, 1.0, by = 0.25)

#rysowanie wykresu temperaturowego ze znalezionym rozwiazaniem
showFunctionContourWithResult <- function(x1, x2, f, GA) {
  filled.contour(x1,
                 x2,
                 f,
                 color.palette = bl2gr.colors,
                 plot.axes = {
                   axis(1)
                   axis(2)

                   points(
                     GA@solution[, 1],
                     GA@solution[, 2],
                     pch = 3,
                     cex = 2,
                     col = "white",
                     lwd = 2
                   )
                 })
}

#generacja kodu latex do wstawienia wykresow
getPlotName <- function(...) {
  sprintf(
    "\\clearpage\\begin{figure}[!htbp]
```

```
    \\centering
    \\mbox{
    \\subfigure{
    \\includegraphics[width=3in]{{{inc/results/%s}}}\\quad
    }
    \\subfigure{
    \\includegraphics[width=3in]{{{inc/results/%s}}}\\quad
    }
    }
    \\caption{%s %s p%s i%s c%s m%s e%s}
    \\end{figure}", ... )
}

#minimalizacja GA oraz zapis wykresow
calculateGA <-
  function(functionName,
           popSize,
           iterationSize,
           elitsimPercentage,
           pcrossover,
           pmutation) {
    #wrapper funkcji
    testFunctionWrapper <- function(x1, x2)
    {
      goTest(par = c(x1, x2) , fnName = functionName)
    }

    customMutation <- function(object, parent)
    {
      mod <- parent %% 2
      if(mod == 0){
        return (parent * 2)
      } else {
        return (parent / 2) + 1
      }
    }

    customCrossover <- function(object, parents)
    {
      parent_1 <- parents[[1]]
      parent_2 <- parents[[2]]
      wektor_1 <- c(parent_1, parent_2)
      fitness = testFunctionWrapper(parent_1, parent_2)

      tmp_parent_1 <- parents[[1]] + runif(1, -1, 1)
      tmp_parent_2 <- parents[[2]] + runif(1, 1, -1)
      tmp_wektor_1 <- c(parent_1, parent_2)
      tmp_fitness = testFunctionWrapper(parent_1, parent_2)

      if(tmp_fitness > fitness){
        return (list(children=matrix(tmp_wektor_1), fitness=tmp_fitness))
      }
      return (list(children=matrix(wektor_1), fitness=fitness))
    }

    #rozpatrywana przestrzen
```

```
x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)
f <- outer(x1, x2, Vectorize(testFunctionWrapper))

#obliczenie liczby populacji elitarnej
elitsim = round(popSize * elitsimPercentage)

#6 wartosci (kolumn) x liczba iteracji (wiersze)
tmpGASummary <- matrix(0, iterationSize, 6)

#ilosc uruchomien testu
for (test in 1:testInstances) {
  #minimizacja GA:
  GA <- ga(
    type = "real-valued",
    fitness = function(x)
      - Vectorize(testFunctionWrapper(x[1], x[2])),
    # uwaga na minusa, bo szukamy glob. minimum
    min = c(-5.12, -5.12),
    #mutation = customMutation,
    crossover = customCrossover,
    #rozpatrywana przestrzen
    max = c(5.12, 5.12),
    maxiter = iterationSize,
    run = iterationSize,
    monitor = FALSE #wylaczenie logowania
  )
  #sumowanie rozwiazan
  tmpGASummary <- GA@summary + tmpGASummary
}
#wyznaczenie sredniej arytmetycznej rozwiazan
tmpGASummary <- tmpGASummary / testInstances

#nazwa pliku z uzytymi parametrami
name <- sprintf(
  "%s-p%03d-i%03d-c%.2f-m%.2f-e%.2f",
  functionName,
  popSize,
  iterationSize,
  pcrossover,
  pmutation,
  elitsimPercentage
)

#nazwa wykresu
filenamePlot = sprintf("generations-%s.jpg", name)
max <- tmpGASummary[, 1]
mean <- tmpGASummary[, 2]
median <- tmpGASummary[, 4]
min <- tmpGASummary[, 6]

#zapis wykresu
jpeg(file = sprintf("%s%s", path, filenamePlot))

#zakres y dla rysowanego wykresu
minPlot <- min(mean) * 0.98
maxPlot <- max(max) * 1.02
```

```
#rysowanie wykresu z zaznaczonymi wartosciami:
#- srednia arytmetyczna rozwiazan,
#- mediana rozwiazan,
#- najlepszym rozwiazaniem
#dla kazdej generacji
plot(
  mean,
  type = "o",
  col = "blue",
  pch = 20,
  ly = 2,
  ann = FALSE,
  ylim = c(minPlot, maxPlot)
)
lines(
  max,
  type = "o",
  col = "green",
  pch = 22,
  lty = 4
)
lines(
  median,
  type = "o",
  col = "red",
  pch = 21,
  lty = 3
)
title(xlab = "Generations")
title(ylab = "Fitness value")
grid()
legend(
  "bottomright",
  c("mean", "best", "median"),
  cex = 0.8,
  col = c("blue", "green", "red"),
  pch = c(20, 22, 21),
  lty = c(2, 4, 3)
)
dev.off()

#zapis wykresu temperaturowego oraz zaznaczenie znalezionego wyniku
fileNameContour = sprintf("result-%s.jpg", name)
jpeg(file = sprintf("%s%s", path, fileNameContour))
showFunctionContourWithResult(x1, x2, f, GA)
dev.off()

plotTitle <- "Test optymalizacji GA"
line = getPlotName(
  filenamePlot,
  fileNameContour,
  plotTitle,
  functionName,
  popSize,
  iterationSize,
```

```
      pcrossover,
      pmutation,
      elitsimPercentage
    )

    #zapis kodu latex do wygenerowanych wykresow
    write(line,
          file = sprintf("%s_latex.txt", path),
          append = TRUE)
  }

#uruchomienie testow dla roznych parametrow dla danej funkcji z argumentu
invokeTestsWithFunction <- function(functionName) {
  #zmiana wartosci populacji elitarnej

    calculateGA(
      functionName,
      defaultPopSize,
      defaultIterationSize,
      defaultElitePopulation,
      defaultCrossover,
      defaultMutation
    )
}

#START
for (fnName in fnNames) {
  print(sprintf("testing_with_function_%s", fnName))
  invokeTestsWithFunction(fnName)
  print(sprintf("tests_end_for_function_%s", fnName))
}

#https://cran.r-project.org/web/packages/GA/vignettes/GA.html
#install.packages("GA");# do instalacji biblioteki GA
#install.packages("globalOptTests")
#install.packages("TSP")
library(GA)
library(globalOptTests)
library(TSP)
library(igraph)


#domyslne wartosci GA
defaultPopSize = 50
defaultIterationSize = 100
defaultCrossover = 0.8
defaultMutation = 0.1

#domyslne wartosci HGA
defaultMethod <- "L-BFGS-B";
defaultPoptim <- 0.05
defaultPressel <- 0.5

#badane parametry HGA
poptims = seq(0.05, 1, 0.2)
pressels = seq(0, 1, 0.25)
```

```
#badane parametry GA
populationSizes = seq(50, 250, by = 50)
iterSizes = seq(50, 250, by = 50)
crossoverSizes = seq(0, 1.0, by = 0.25)
mutationSizes = seq(0, 1.0, by = 0.25)

#ilosc przebiegow
testInstances = 1

#sciezka zapisu
path = "~/Desktop/ga.nosync/"

# obliczenie calkowitej dlugoeci
tourLength <- function(tour, distMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[, 2:1]
  sum(distMatrix[route])
}

# odwrotnoec calkowitej odlegloeci
tpsFitness <- function(tour, ...) 1 / tourLength(tour, ...)

#funkcja obliczajaca GA i zapisujaca wykresy
calculateGA <- function(fileName, popSize, iterationSize, crossover, mutation) {
  #wczytanie pliku *.tsp
  drill <- read_TSPLIB(sprintf("~/Documents/%s", fileName))

  #konwersja pliku do postaci macierzowej
  D <- as.matrix(drill)

  #przygotowanie macierzy wyjsciowej (do obliczania wartosci srednich)
  fitnessMat <- matrix(0, testInstances, 2)

  # 2-d coordinates
  mds <- cmdscale(D)
  x <- mds[, 1]
  y <- -mds[, 2]
  n <- length(x)

  #petla odpowiedzialna za ilosc przebiegow funkcji GA
  for (instanceIndex in seq(1, testInstances)) {
    GA.rep <-
      ga(
        type = "permutation",
        fitness = tpsFitness,
        distMatrix = D,
        min = 1,
        max = nrow(D),
        popSize = popSize,
        keepBest = TRUE,
        pmutation = mutation,
        pcrossover = crossover,
        maxiter = iterationSize,
        run = iterationSize,
        monitor = FALSE
```

```
    )

    #dodawanie rozwiazan do macierzy wynikowej
    fitnessMat[instanceIndex, 1] <- GA.rep@summary[GA.rep@iter]
    fitnessMat[instanceIndex, 2] <- GA.rep@summary[GA.rep@iter]
  }

  #generowanie nazwy pliku/wykresu
  name = sprintf("%s-pop-%s-iter-%s-cross-%s", fileName, popSize, iterationSize, crossover,

  #zapis wykresu
  jpeg(file = sprintf("%s%s.jpg", path, name))
  plotName = sprintf("%s%s.jpg", path, fileName)

  #generowane wykresu
  plot(GA.rep, main = name)
  points(rep(50, testInstances), fitnessMat[, 1], pch = 16, col = "lightgrey")
  points(rep(55, testInstances), fitnessMat[, 2], pch = 17, col = "lightblue")
  dev.off()
}

#funkcja obliczajaca HGA i zapisujaca wykresy
calculateHGA <- function(fileName, method, poptim, pressel) {
  drill <- read_TSPLIB(sprintf("~/Documents/%s", fileName))
  D <- as.matrix(drill)
  fitnessMat <- matrix(0, testInstances, 2)



  for (instanceIndex in seq(1, testInstances)) {

    optimArgs = list(method = defaultMethod,
                     poptim = poptim,
                     pressel = pressel,
                     control = list(fnscale = -1, maxit = 100))

    # run a HGA algorithm
    GA.rep <-
      ga(
        type = "permutation",
        fitness = tpsFitness,
        distMatrix = D,
        min = 1,
        max = nrow(D),
        monitor = FALSE,
        optimArgs = optimArgs
      )

    fitnessMat[instanceIndex, 1] <- GA.rep@summary[GA.rep@iter]
    fitnessMat[instanceIndex, 2] <- GA.rep@summary[GA.rep@iter]
  }


  #generowanie nazwy pliku/wykresu
  name = sprintf("hga-%s-method-%s-poptim-%s-pressel-%s", fileName, method, poptim, pressel)
```

```
  #zapis wykresu
  jpeg(file = sprintf("%s%s.jpg", path, name))

  #generowane wykresu
  plot(GA.rep, main = name)
  points(rep(50, testInstances), fitnessMat[, 1], pch = 16, col = "lightgrey")
  points(rep(55, testInstances), fitnessMat[, 2], pch = 17, col = "lightblue")
  dev.off()
}

#funkcja uruchamiajace GA dla roznych parametrow
invoke <- function(fileName) {

  #zmiana wartosci krzyzowania
  for (pcrossover in crossoverSizes) {
    calculateGA(
      fileName,
      defaultPopSize,
      defaultIterationSize,
      pcrossover,
      defaultMutation
    )
  }

  #zmiana wartosci liczby iteracji
  for (iterationSize in iterSizes) {
    calculateGA(
      fileName,
      defaultPopSize,
      iterationSize,
      defaultCrossover,
      defaultMutation
    )
  }

  #zmiana liczby popopulacji
  for (popSize in populationSizes) {
    calculateGA(
      fileName,
      popSize,
      defaultIterationSize,
      defaultCrossover,
      defaultMutation
    )
  }
}

#funkcja uruchamiajace HGA dla roznych parametrow
invokeHGA <- function(fileName) {

  #zmiana wartosci prawdopodobienstwa przeszukiwania lokalnego
  for (poptim in poptims) {
    calculateHGA(
      fileName,
      defaultMethod,
      poptim,
```

```
      defaultPressel
    )
  }

  #zmiana wartosci selective pressure
  for (pressel in pressels) {
    calculateHGA(
      fileName,
      defaultMethod,
      defaultPoptim,
      pressel
    )
  }
}

invokeHGA('bays29.tsp')
invoke('bays29.tsp')
invoke('gr17.tsp')
invoke('gr120.tsp')

#https://cran.r-project.org/web/packages/GA/vignettes/GA.html
#install.packages("GA");# do instalacji biblioteki GA
#install.packages("globalOptTests")
#install.packages("TSP")
library(GA)
library(globalOptTests)
library(TSP)

drill <- read_TSPLIB("~/Documents/gr120.tsp")
D <- as.matrix(drill)

# given a tour, calculate the total distance
tourLength <- function(tour, distMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[, 2:1]
  sum(distMatrix[route])
}

# inverse of thetotal distance is the fitness
tpsFitness <- function(tour, ...) 1/tourLength(tour, ...)

########################################
## crossover/mutation
crossoverSizes = seq(0, 1.0, by = 0.1)
mutationSizes = seq(0, 1.0, by = 0.1)
crossoverSizesLength <- length(crossoverSizes)
mutationSizesLength <- length(mutationSizes)

tempMat <- matrix(0, crossoverSizesLength, mutationSizesLength)

for(crossover in seq(1,crossoverSizesLength )){
  for(mutation in seq(1,mutationSizesLength )){
    B <- 10
    fitnessMat <- matrix(0, B, 2)
    for (b in seq(1, B)) {
      # run a GA algorithm
```

```
      GA.rep <- ga(type = "permutation", fitness = tpsFitness, distMatrix = D,
                   min = 1, max = nrow(D), popSize = 10, maxiter = 50, run = 100,
                   keepBest=TRUE,
                   pmutation = mutationSizes[mutation],
                   pcrossover = crossoverSizes[crossover],
                   monitor = NULL)

      fitnessMat[b, 1] <- GA.rep@summary[GA.rep@iter]
      fitnessMat[b, 2] <- GA.rep@summary[GA.rep@iter]
    }

    tempMat[crossover,mutation] <- GA.rep@fitnessValue
  }
}

filled.contour(crossoverSizes,
               mutationSizes,
               tempMat,
               color.palette = bl2gr.colors,
               xlab="Crossover Probability",
               ylab="Mutation Probability",
               plot.axes = {
                 axis(1)
                 axis(2)
               })


plot(GA.rep, main = "Best and Avg at 50th iteration over 100 simulations")
points(rep(50, B), fitnessMat[, 1], pch = 16, col = "lightgrey")
points(rep(55, B), fitnessMat[, 2], pch = 17, col = "lightblue")

#https://cran.r-project.org/web/packages/GA/vignettes/GA.html
#install.packages("GA");# do instalacji biblioteki GA
#install.packages("globalOptTests")
#install.packages("TSP")
library(GA)
library(globalOptTests)
library(TSP)

drill <- read_TSPLIB("~/Documents/gr17.tsp")
D <- as.matrix(drill)

# given a tour, calculate the total distance
tourLength <- function(tour, distMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[, 2:1]
  sum(distMatrix[route])
}

# inverse of thetotal distance is the fitness
tpsFitness <- function(tour, ...) 1/tourLength(tour, ...)

###################################
## pop size / iteration

populationSizes = seq(50, 250, by = 50)
```

```
iterationSizes = seq(10, 100, by = 20)

populationSizesLength <- length(populationSizes)
iterationSizesLength <- length(iterationSizes)

tempMat <- matrix(0, populationSizesLength, iterationSizesLength)

for(population in seq(1,populationSizesLength )){
  for(iteration in seq(1,iterationSizesLength )){
    B <- 10
    fitnessMat <- matrix(0, B, 2)
    for (b in seq(1, B)) {
      # run a GA algorithm
      GA.rep <- ga(type = "permutation", fitness = tpsFitness, distMatrix = D,
                   min = 1, max = nrow(D), run = 100,
                   keepBest=TRUE,
                   popSize = populationSizes[population],
                   maxiter = iterationSizes[iteration],
                   monitor = NULL)

      fitnessMat[b, 1] <- GA.rep@summary[GA.rep@iter]
      fitnessMat[b, 2] <- GA.rep@summary[GA.rep@iter]
    }

    tempMat[population,iteration] <- GA.rep@fitnessValue
  }
}

filled.contour(populationSizes,
               iterationSizes,
               tempMat,
               color.palette = bl2gr.colors,
               xlab="Population Size",
               ylab="Generation Amount",
               plot.axes = {
                 axis(1)
                 axis(2)
               })

plot(GA.rep, main = "Best and Avg at 50th iteration over 100 simulations")
points(rep(50, B), fitnessMat[, 1], pch = 16, col = "lightgrey")
points(rep(55, B), fitnessMat[, 2], pch = 17, col = "lightblue")
```