

POLITECHNIKA WROCŁAWSKA

LABORATORIUM

INTELIGENCJA OBliczeniowa i JEJ ZASTOSOWANIA

Algorytmy ewolucyjne i hybrydowe

Authors:

Rafał PIENIĄŻEK
Jakub POMYKAŁA

Supervisor:

prof. dr inż. Olgierd UNOLD

22 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 ewoluje zgodnie z regułami selekcji zgodnie z funkcją celu przypisaną do środowiska. Propagowane do kolejnych pokoleń są tylko najbardziej dopasowane osobniki.

2.1 Modyfikowane parametry

W celu badania wpływu zmiennych na zachowanie algorytmu genetycznego modyfikowano następujące parametry:

- **elitarność** - Procentowa wartość populacji, która zostaje przeniesiona do następnego pokolenia bez zmian. Niezerowa wartość gwarantuje, że jakość rozwiązania nie zmniejszy się z pokolenia na pokolenie.
- **mutacja** - Przypadkowa zmiana genomu w algorytmie genetycznym, analogicznie do mutacji biologicznej.
- **krzyżowanie** - Połączenie niektórych (wybranych losowo) genotypów w jeden. Kojarzenie ma sprawić, że potomek dwóch osobników rodzicielskich ma zespół cech, który jest kombinacją ich cech (może się zdarzyć, że tych najlepszych).
- **liczba iteracji** - Ilość pokoleń
- **rozmiar populacji** - Ilość osobników w jednym pokoleniu.

Tabela 1: Wartości modyfikowanych parametrów

Parametr	Wartości
populacja	50, 100, 150, 200, 250
liczba iteracji	50, 100, 150, 200, 250
p. krzyżowania	0, 0.25, 0.5, 0.75, 1
p. mutacji	0, 0.25, 0.5, 0.75, 1
p. populacji elitarnej	0, 0.25, 0.5, 0.75, 1

2.2 Zastosowane narzędzia implementacji

2.2.1 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.

2.2.2 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.

2.2.3 Pakiet globalOpts

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

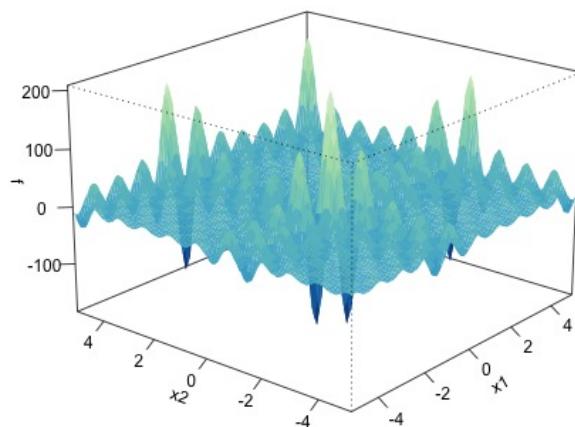
3 Funkcja Schuberta

3.1 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

3.2 Wykres w ustalonym przedziale zmiennych

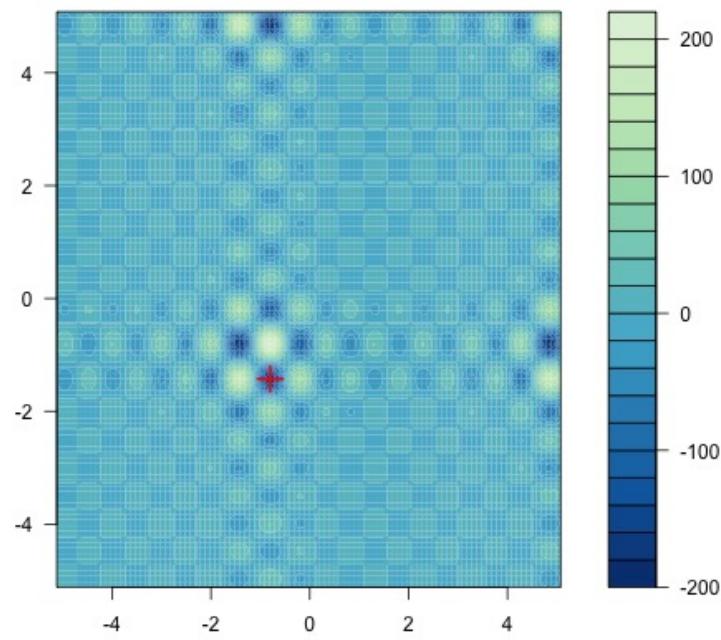


Rysunek 2: Wykres funkcji Schuberta

3.3 Ekstremum globalne

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

Rysunek 3: Minimum globalne dla funkcji Schuberta

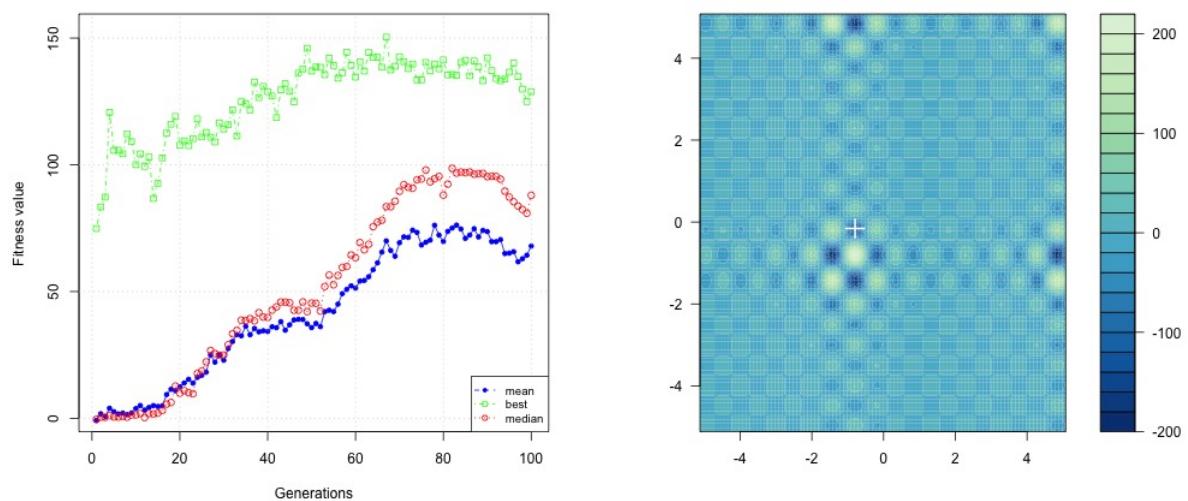


Rysunek 4: Minimum globalne dla funkcji Schuberta

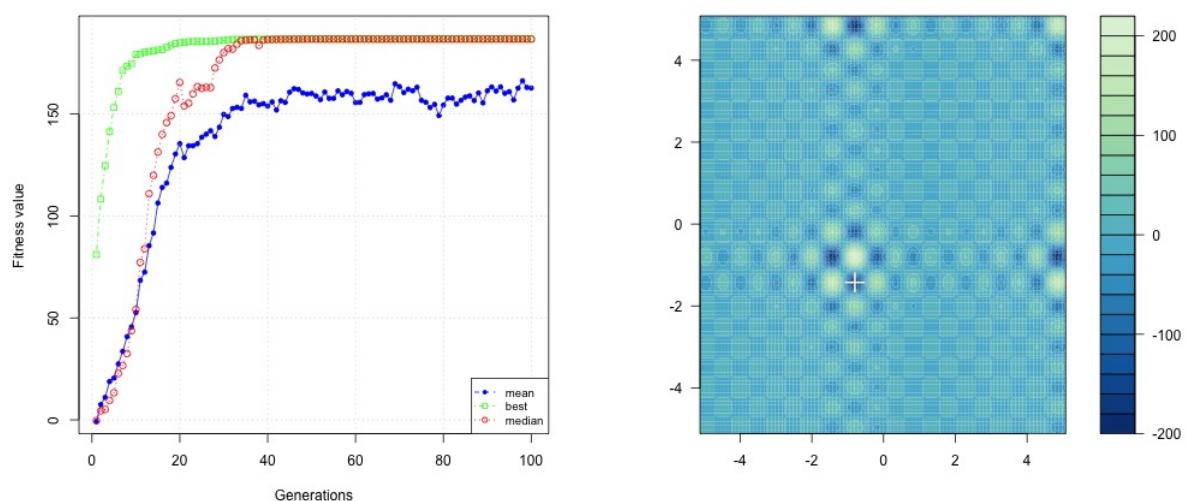
3.4 Optymalizacja poszukiwania ekstremum globalnego

3.4.1 Modyfikacja parametru elitarności

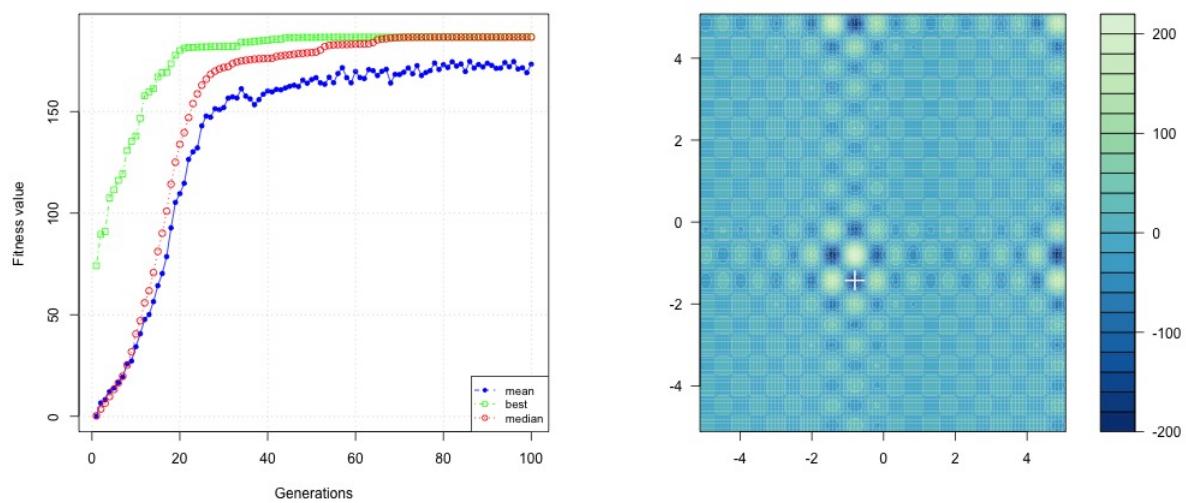
W przypadku zwiększenia wartości elitarności populacji średnie wartości odchyleń zmniejszają się. W przypadku większej elitarności algorytm szybciej znajduje optymalne rozwiązanie. Jeżeli elitarność populacji jest równa 1, to algorytm stochastycznie wylosował jeden zestaw rozwiązań, który z powodu braku ewolucji nie zbliżył się do rozwiązania optymalnego.



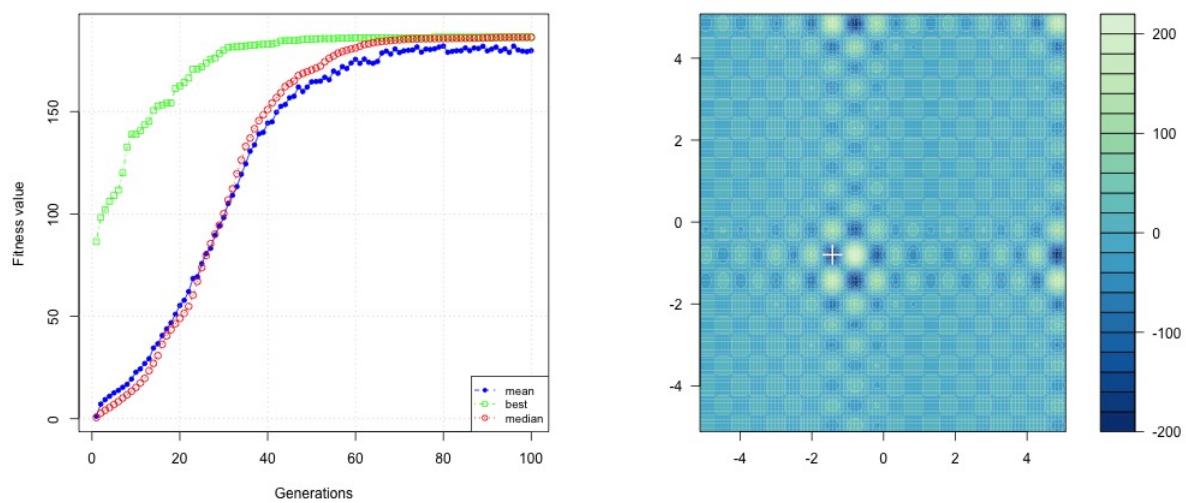
Rysunek 5: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e0



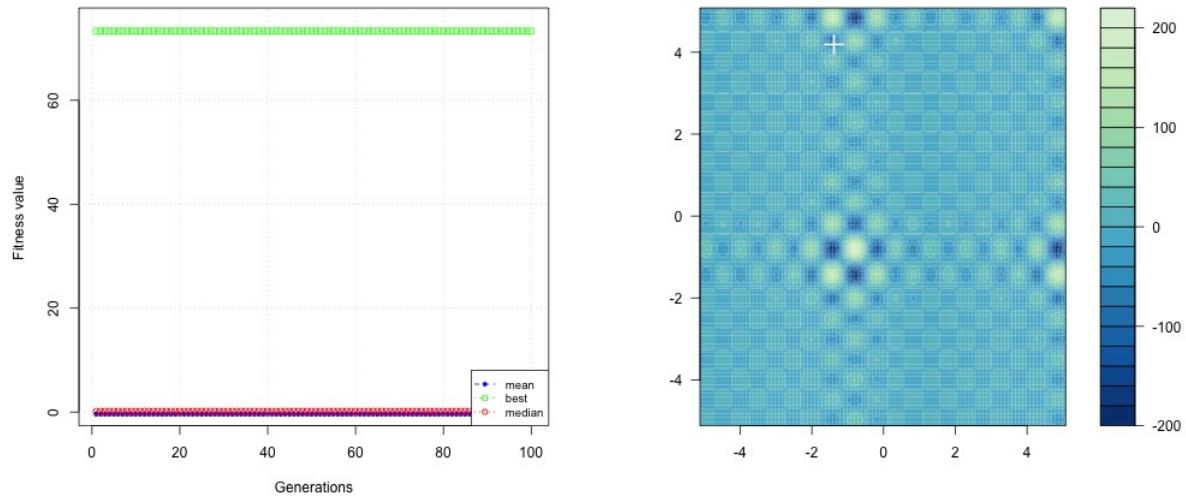
Rysunek 6: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e0.25



Rysunek 7: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e0.5



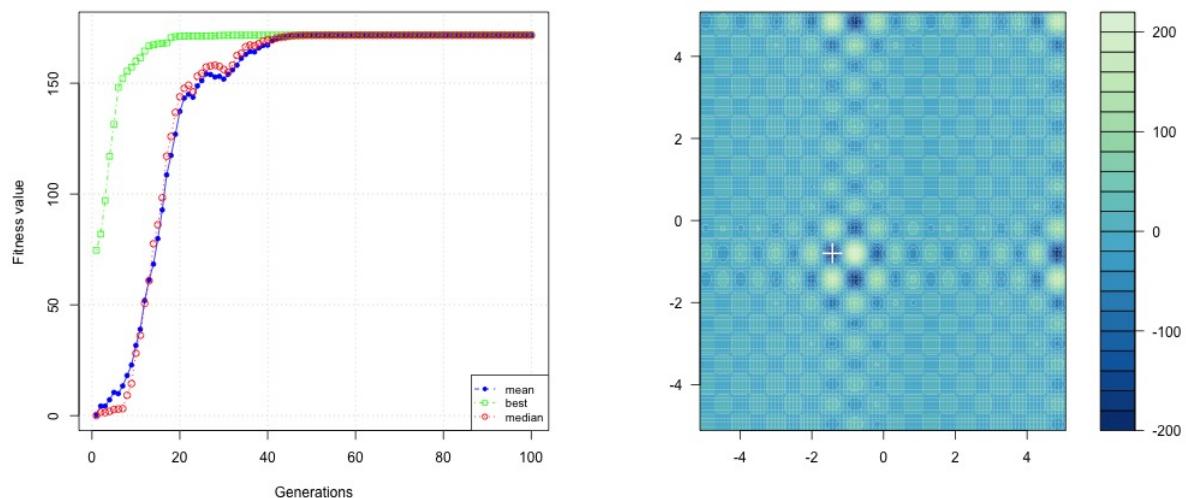
Rysunek 8: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e0.75



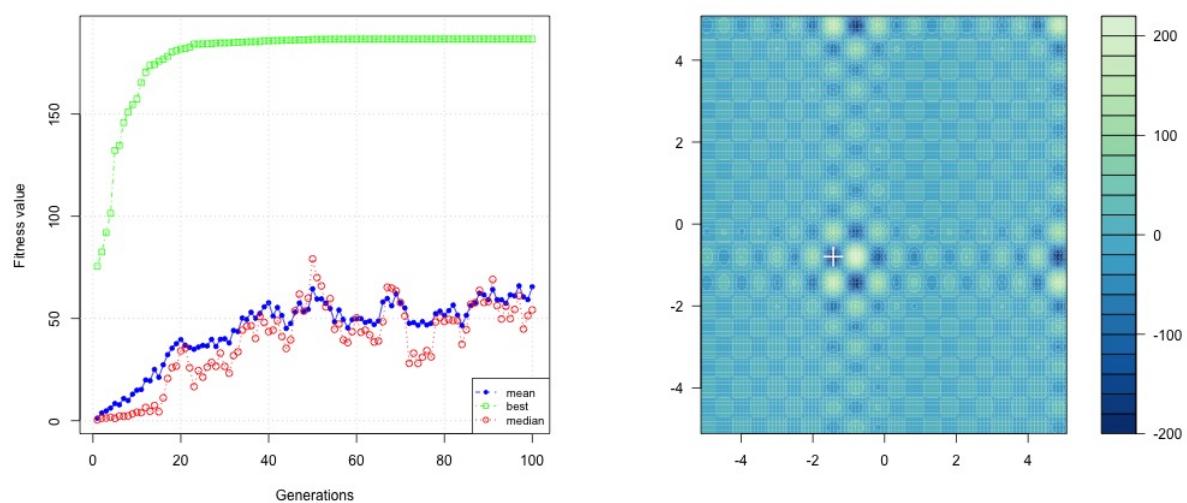
Rysunek 9: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e1

3.4.2 Modyfikacja parametru mutacji

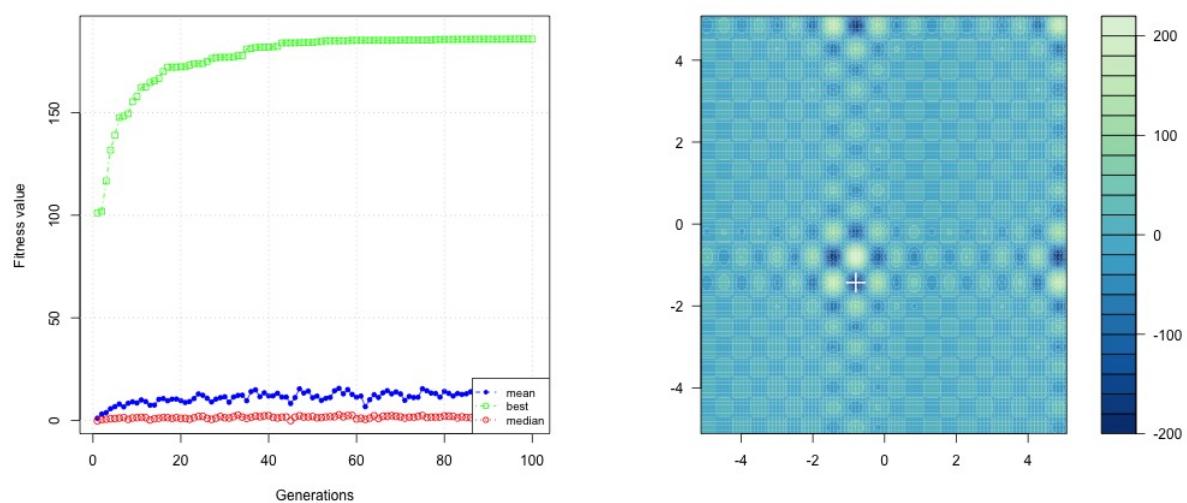
W przypadku zwiększenia wartości mutacji zarówno średnia jak i mediana zmniejszają się. Algorytm zawsze znajduje po podobnej liczbie populacji rozwiązanie optymalne.



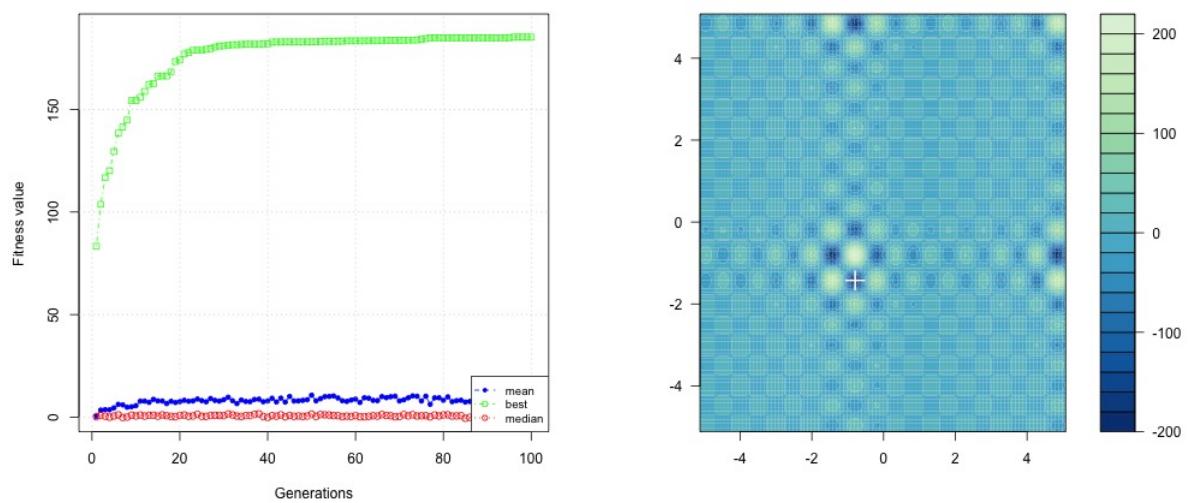
Rysunek 10: Test optymalizacji GA Schubert p50 i100 c0.8 m0 e0.05



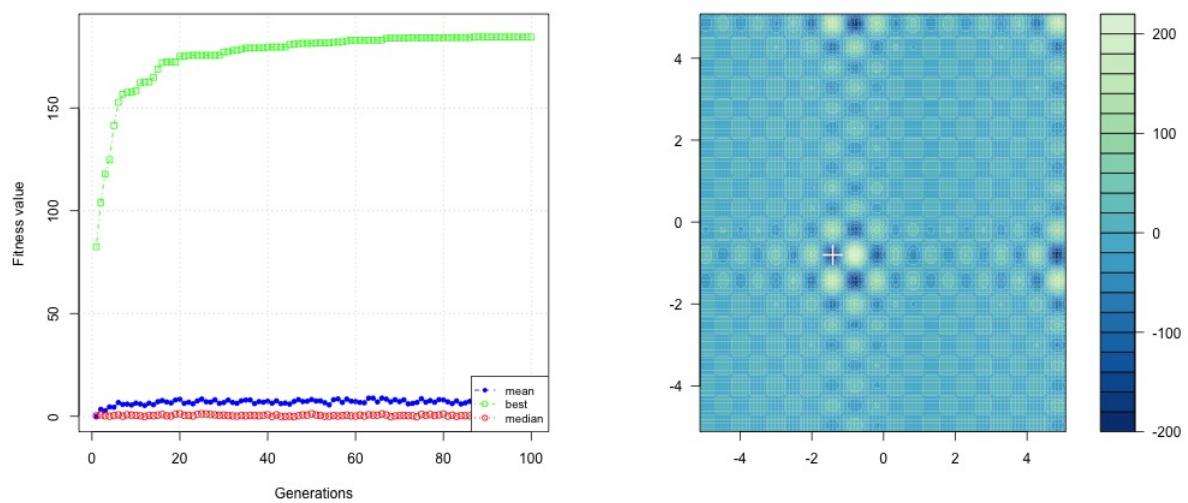
Rysunek 11: Test optymalizacji GA Schubert p50 i100 c0.8 m0.25 e0.05



Rysunek 12: Test optymalizacji GA Schubert p50 i100 c0.8 m0.5 e0.05



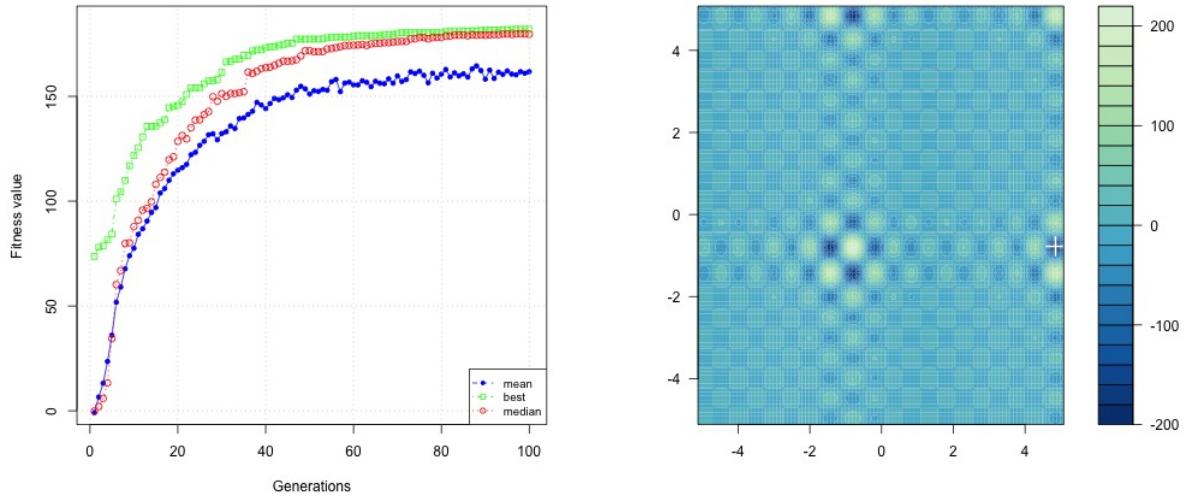
Rysunek 13: Test optymalizacji GA Schubert p50 i100 c0.8 m0.75 e0.05



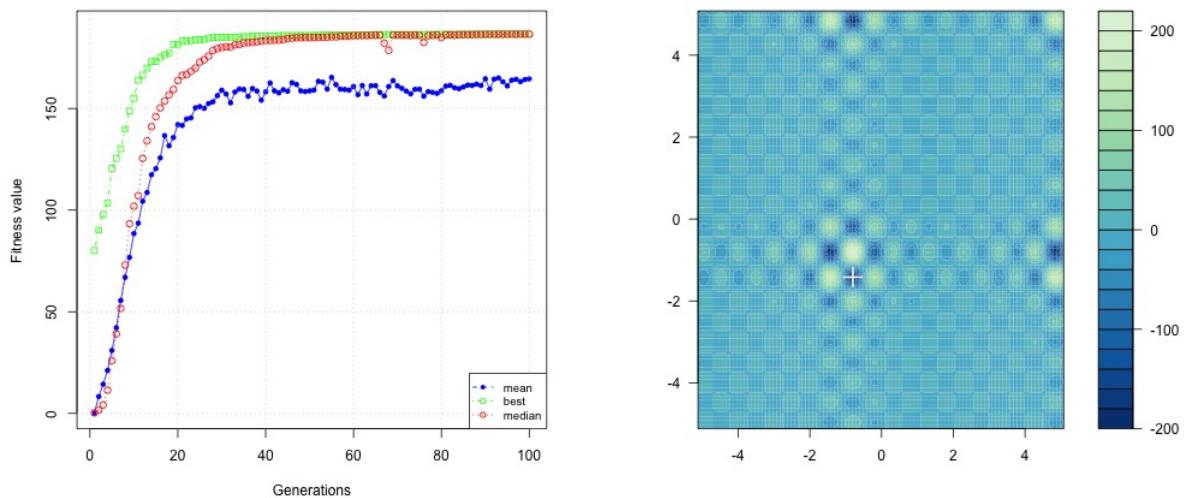
Rysunek 14: Test optymalizacji GA Schubert p50 i100 c0.8 m1 e0.05

3.4.3 Modyfikacja parametru krzyżowania

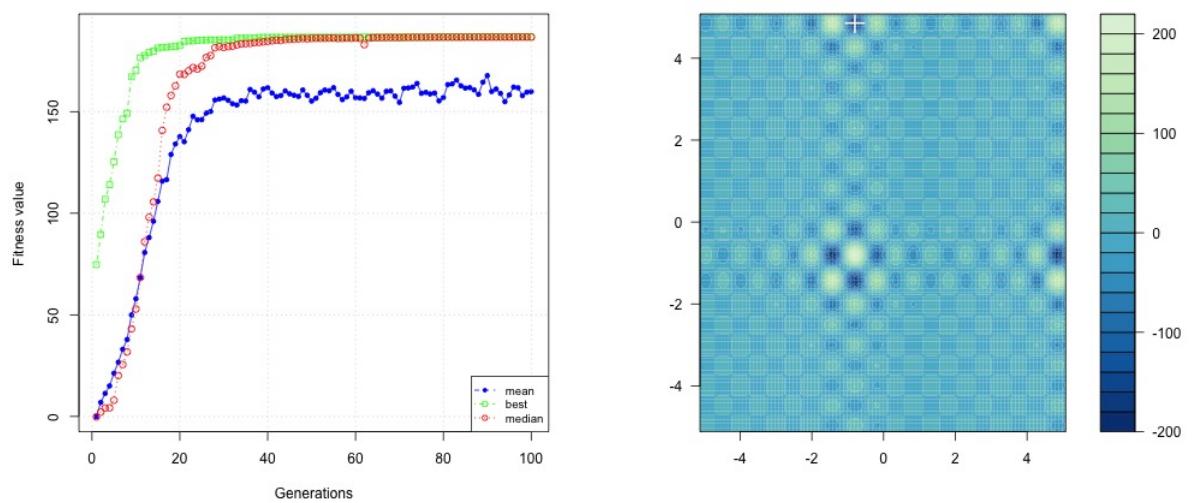
W przypadku zwiększenia parametru krzyżowania algorytm szybciej znajduje lokalne minimum, ale nie zawsze jest to ekstremum globalne. Wartości średniej i mediany pozostają na podobnym poziomie niezależnie od parametrów.



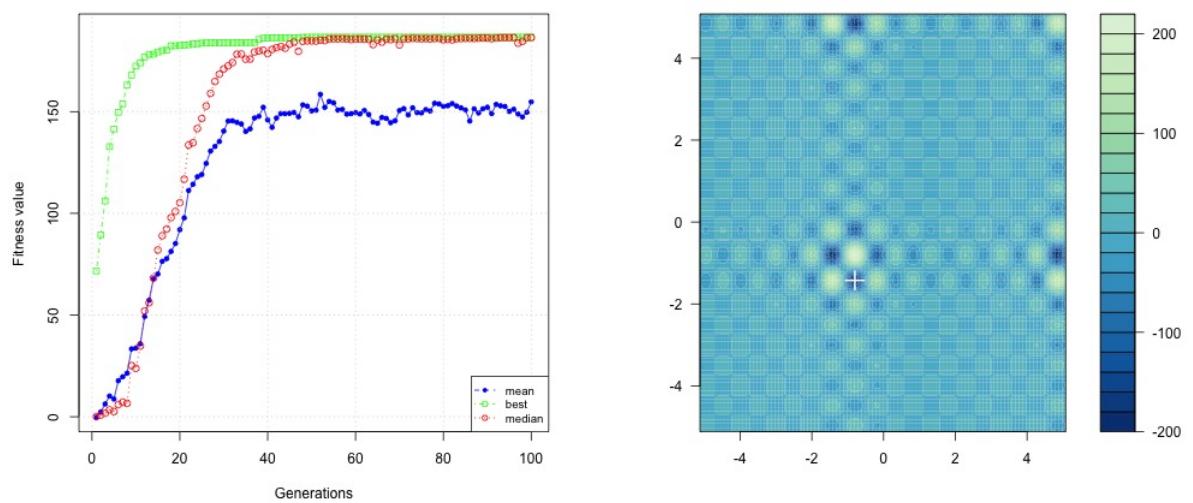
Rysunek 15: Test optymalizacji GA Schubert p50 i100 c0 m0.1 e0.05



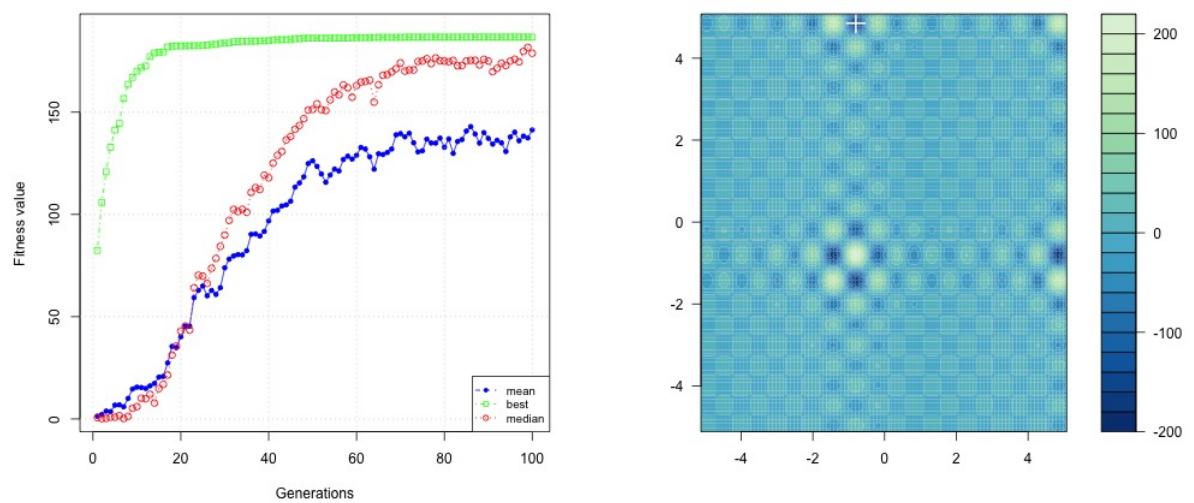
Rysunek 16: Test optymalizacji GA Schubert p50 i100 c0.25 m0.1 e0.05



Rysunek 17: Test optymalizacji GA Schubert p50 i100 c0.5 m0.1 e0.05



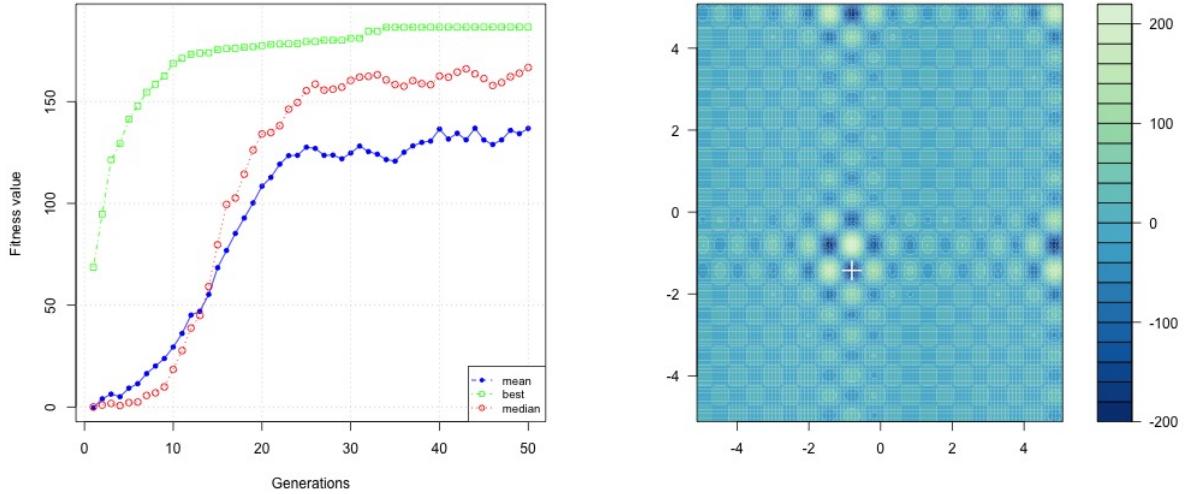
Rysunek 18: Test optymalizacji GA Schubert p50 i100 c0.75 m0.1 e0.05



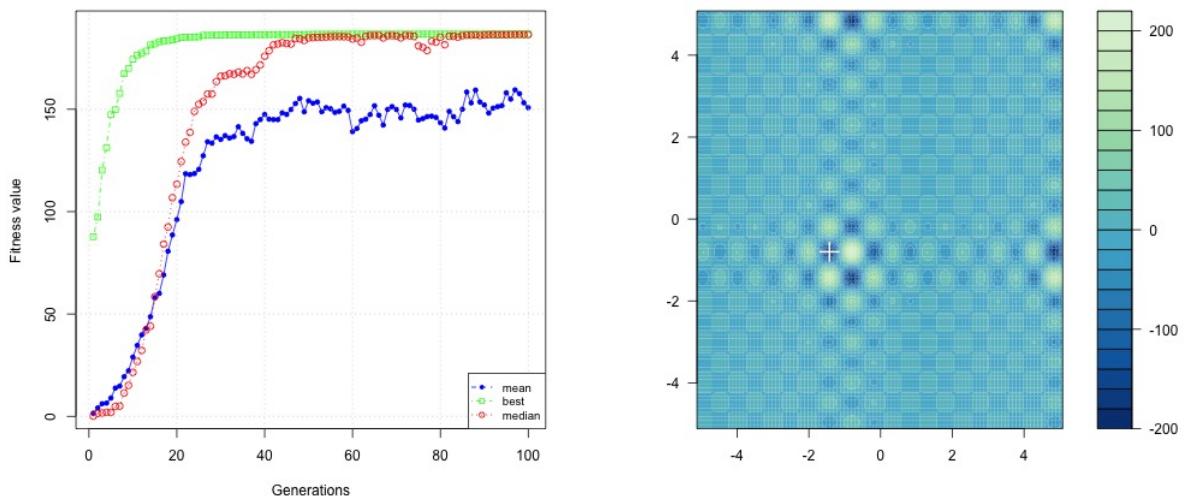
Rysunek 19: Test optymalizacji GA Schubert p50 i100 c1 m0.1 e0.05

3.4.4 Modyfikacja parametru liczby iteracji

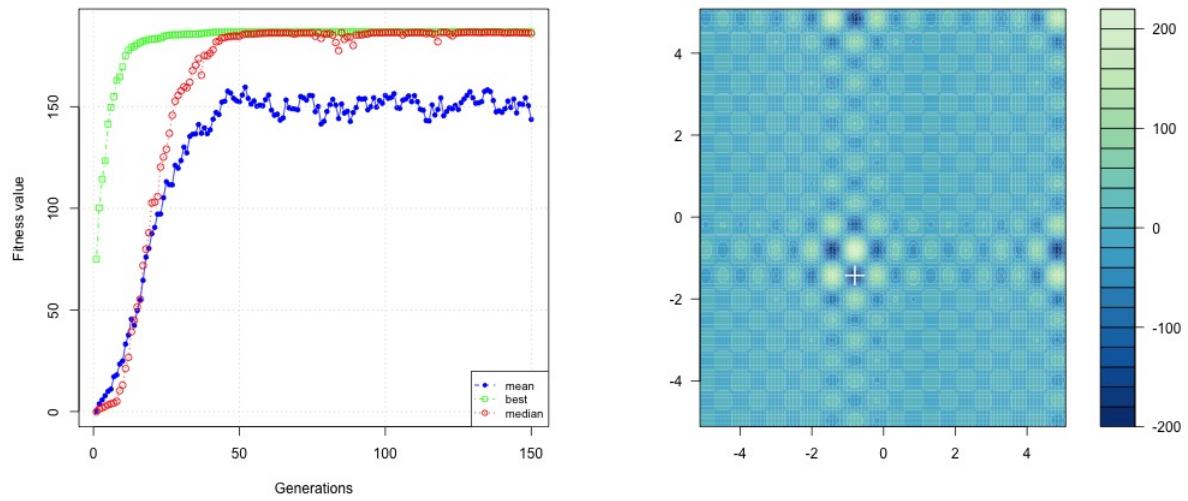
W przypadku zwiększenia ilości iteracji zarówno średnia jak i mediana pozostają na podobnym poziomie. Zmiana parametru liczby iteracji nie wpływa również znacząco na wyniki.



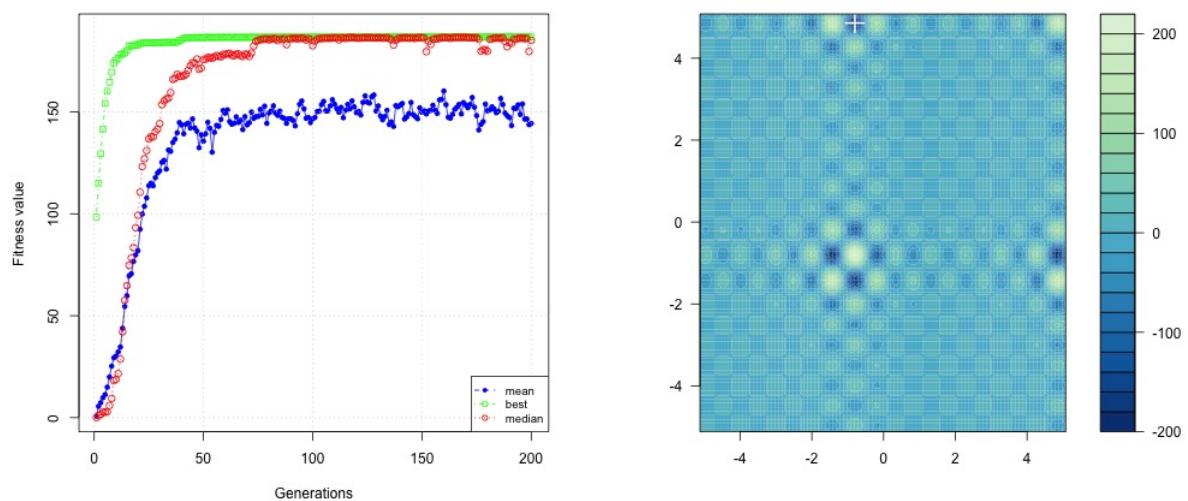
Rysunek 20: Test optymalizacji GA Schubert p50 i50 c0.8 m0.1 e0.05



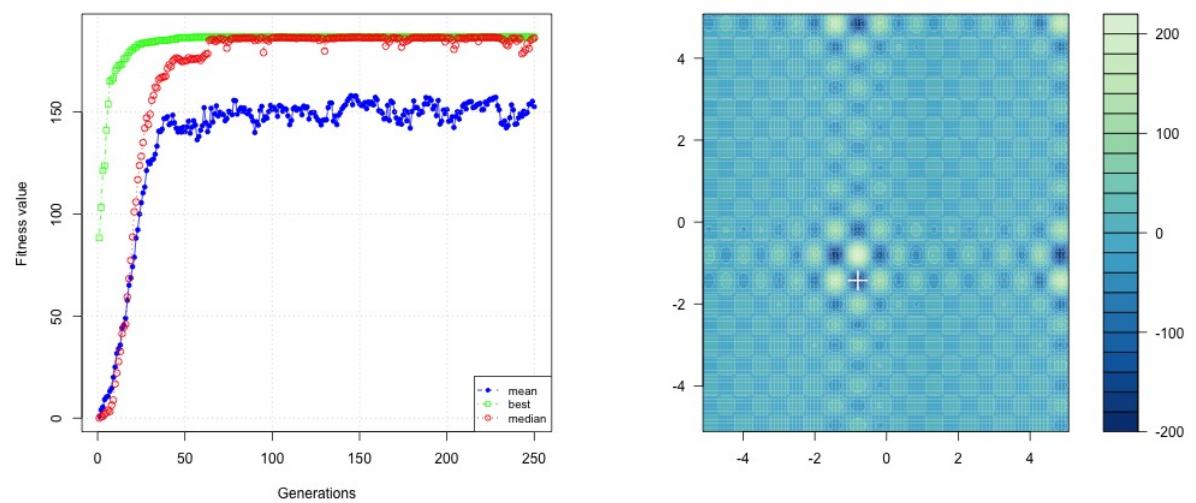
Rysunek 21: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e0.05



Rysunek 22: Test optymalizacji GA Schubert p50 i150 c0.8 m0.1 e0.05



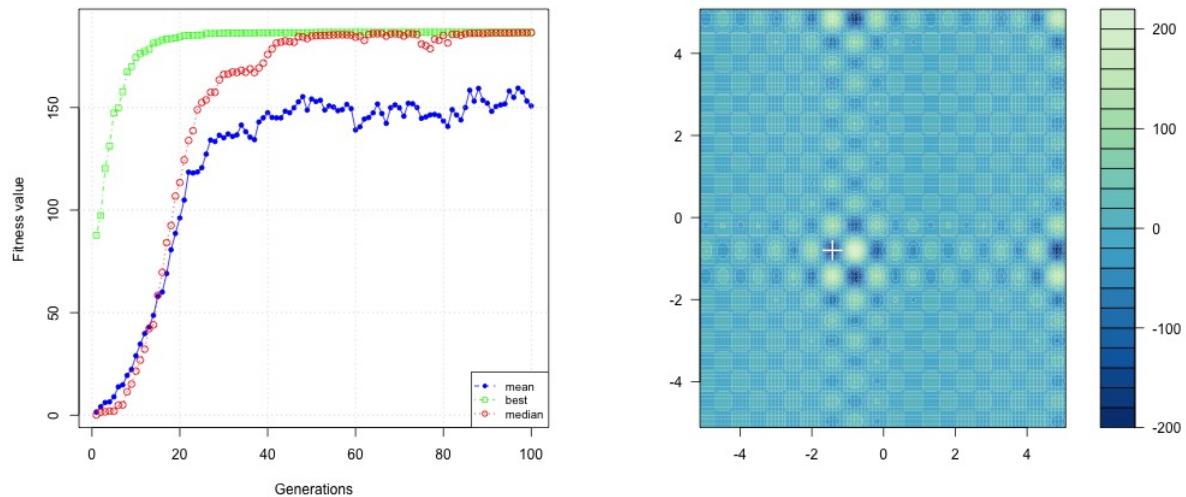
Rysunek 23: Test optymalizacji GA Schubert p50 i200 c0.8 m0.1 e0.05



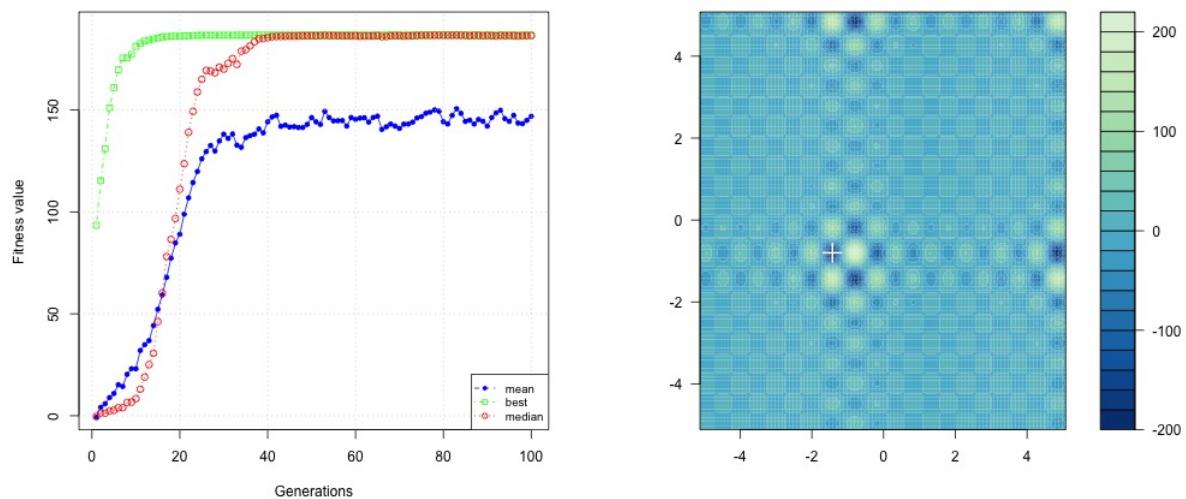
Rysunek 24: Test optymalizacji GA Schubert p50 i250 c0.8 m0.1 e0.05

3.4.5 Modyfikacja parametru rozmiaru populacji

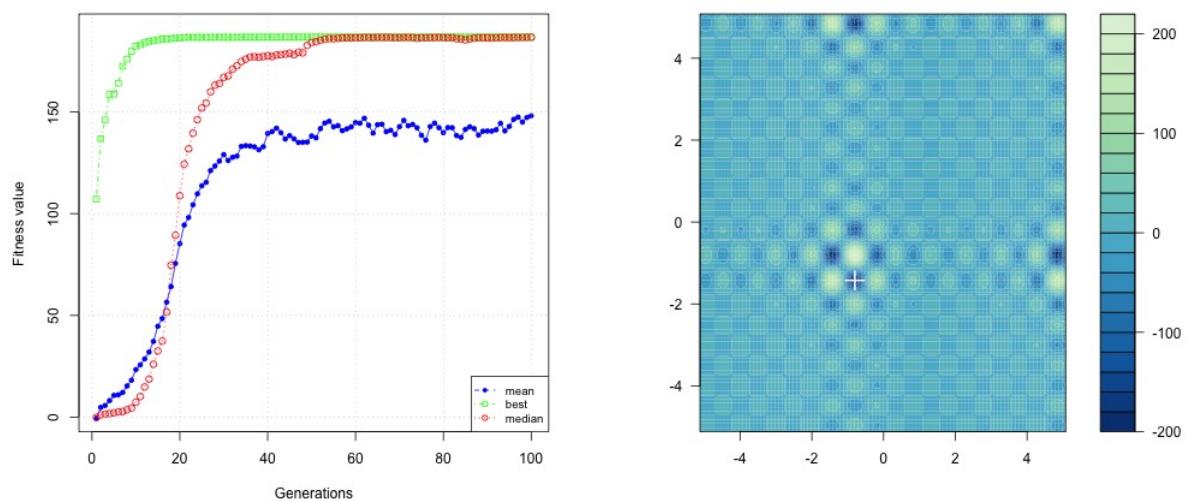
Gdy parametr rozmiaru populacji jest ustawiony na wartość populacji 250 można zaobserwować małą anomalię średniej i mediany w okolicach 50-60 pokolenia. Anomalie średniej i mediany są skorelowane ze sobą, jednak nie wpływają na znalezione rozwiązanie. Niezależnie od rozmiaru parametru algorytm znajduje minimum globalne przy podobnej ilości pokoleń.



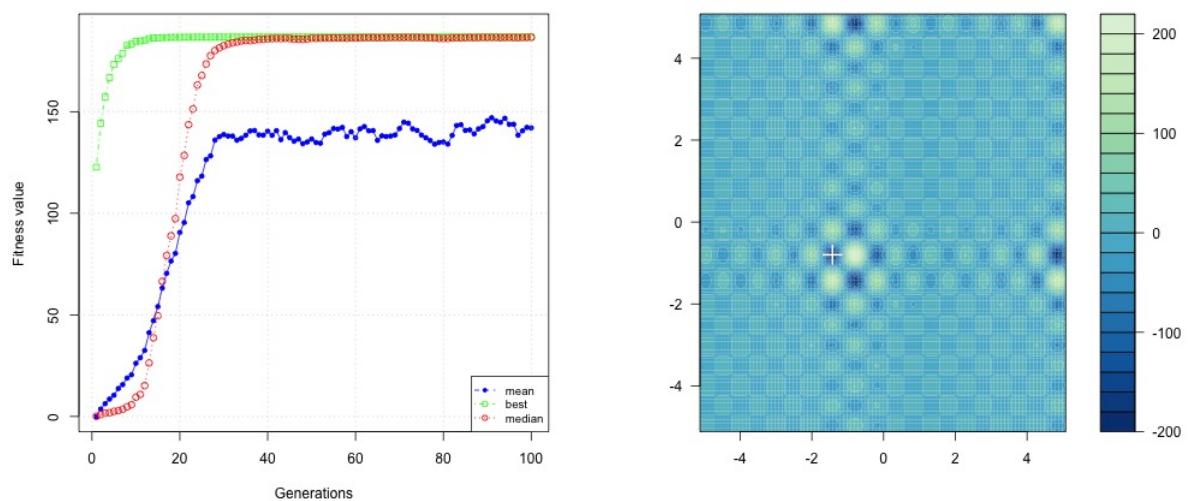
Rysunek 25: Test optymalizacji GA Schubert p50 i100 c0.8 m0.1 e0.05



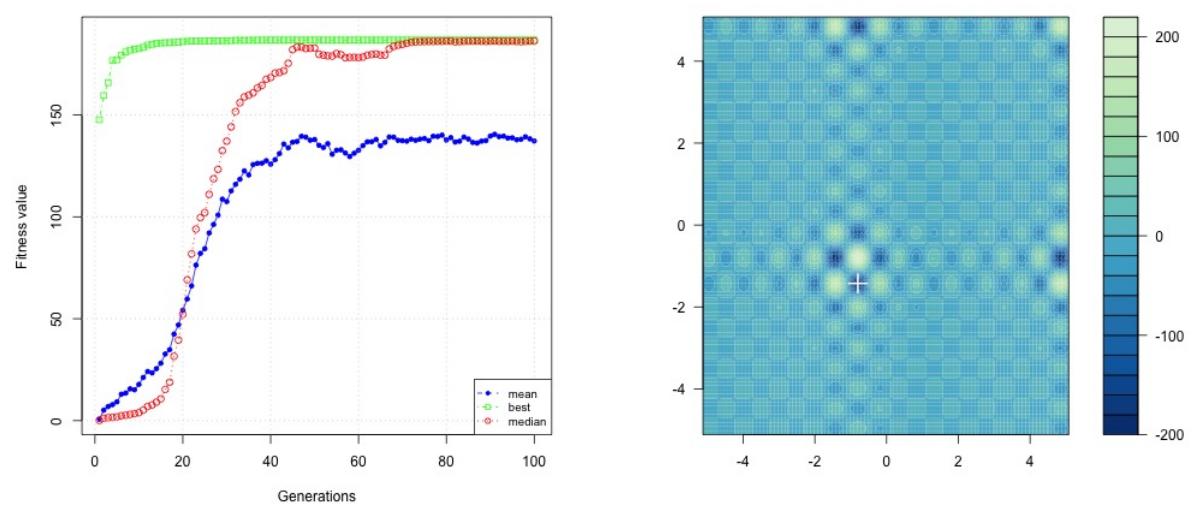
Rysunek 26: Test optymalizacji GA Schubert p100 i100 c0.8 m0.1 e0.05



Rysunek 27: Test optymalizacji GA Schubert p150 i100 c0.8 m0.1 e0.05



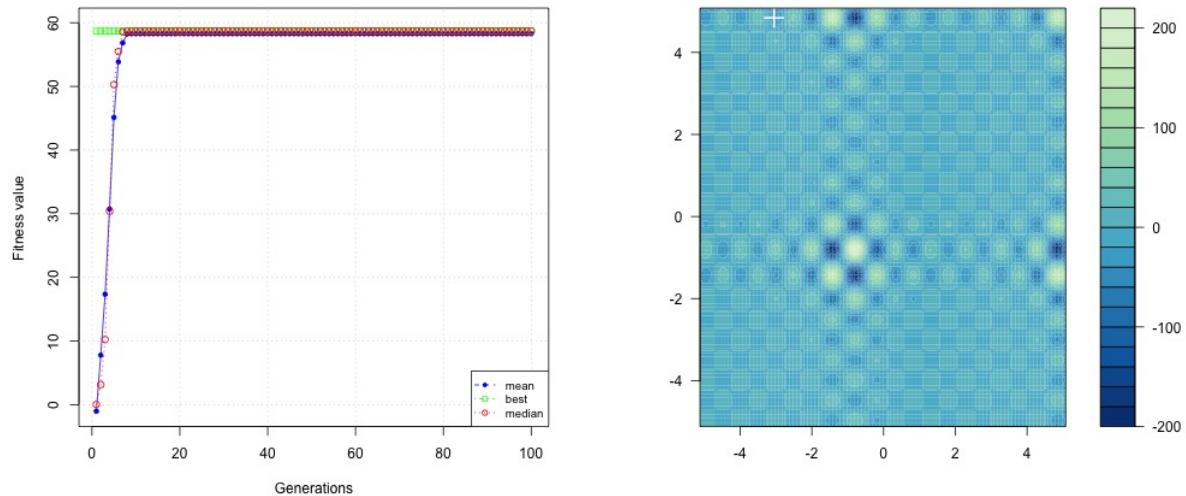
Rysunek 28: Test optymalizacji GA Schubert p200 i100 c0.8 m0.1 e0.05



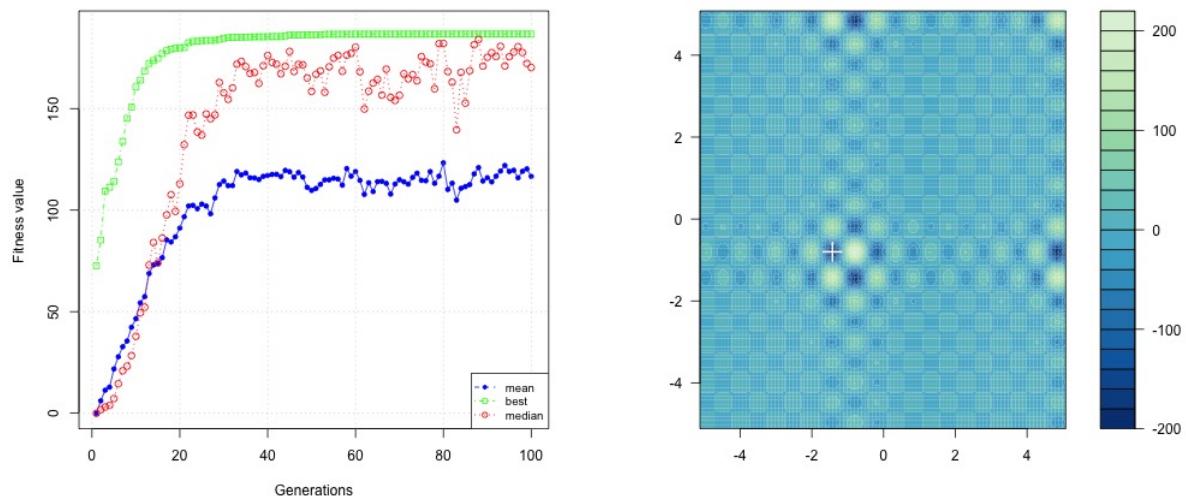
Rysunek 29: Test optymalizacji GA Schubert p250 i100 c0.8 m0.1 e0.05

3.5 Jednoczesna modyfikacja parametru krzyżowania i mutacji

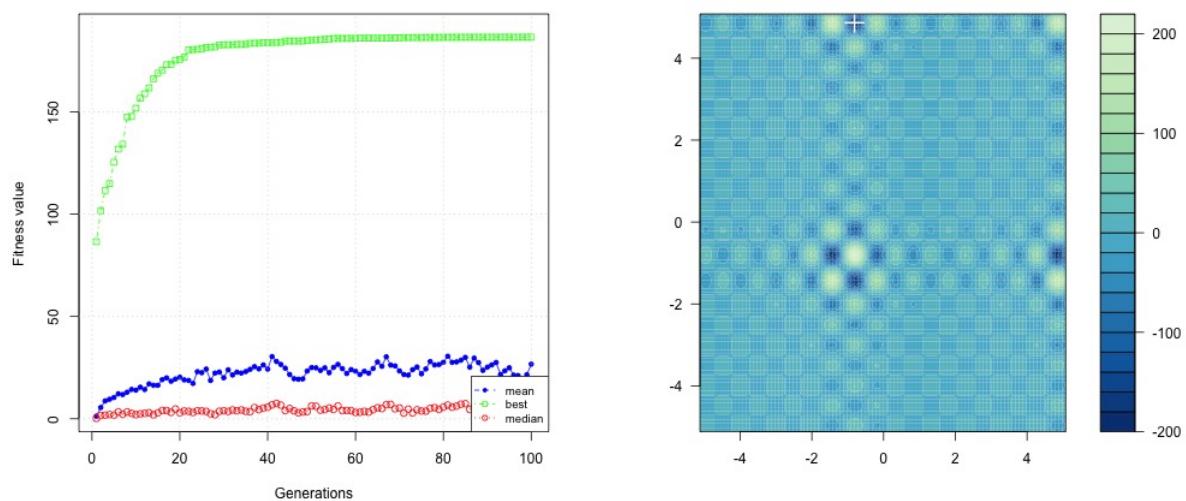
W przypadku jednoczesnej modyfikacji parametru krzyżowania i mutacji wyniki mają charakter losowy, lecz skupiają się wokół minimum lokalnych. Wraz ze wzrostem obydwu parametrów wartości średniej i mediany maleją.



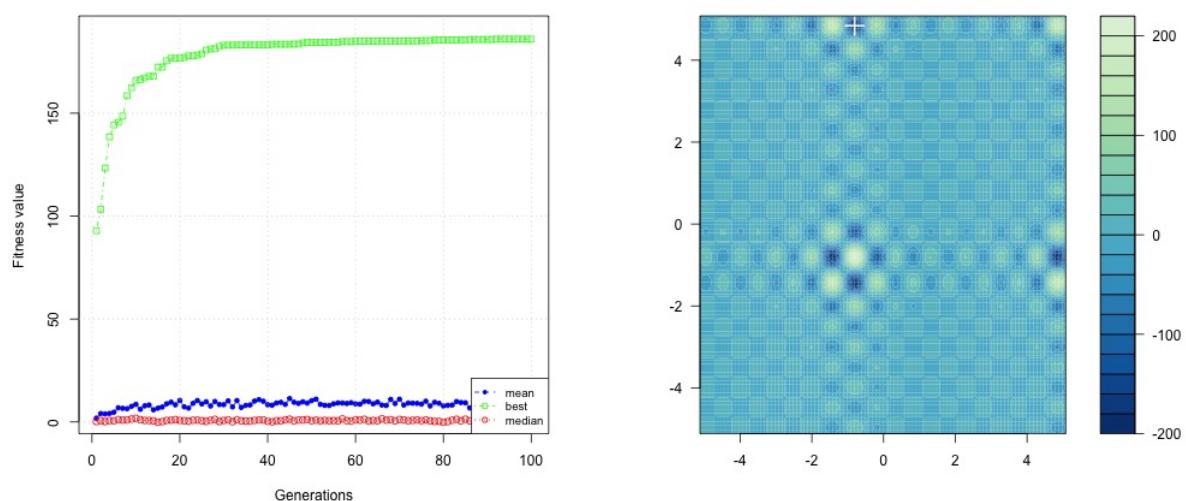
Rysunek 30: Test optymalizacji GA Schubert p050 i100 c0.0 m0.0 e0.05



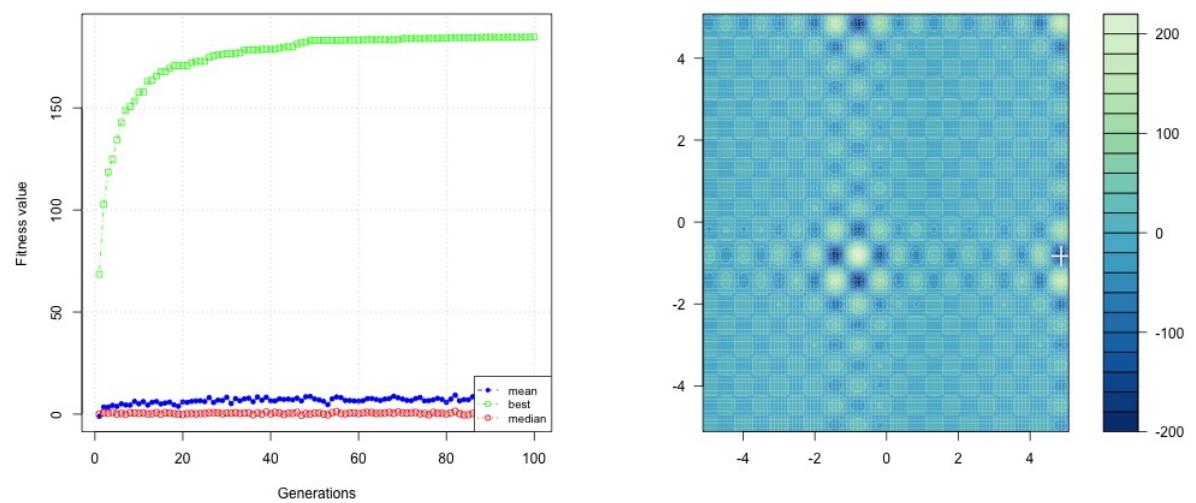
Rysunek 31: Test optymalizacji GA Schubert p050 i100 c0.25 m0.25 e0.05



Rysunek 32: Test optymalizacji GA Schubert p050 i100 c0.50 m0.50 e0.05



Rysunek 33: Test optymalizacji GA Schubert p050 i100 c0.75 m0.75 e0.05



Rysunek 34: Test optymalizacji GA Schubert p050 i100 c1.00 m1.00 e0.05

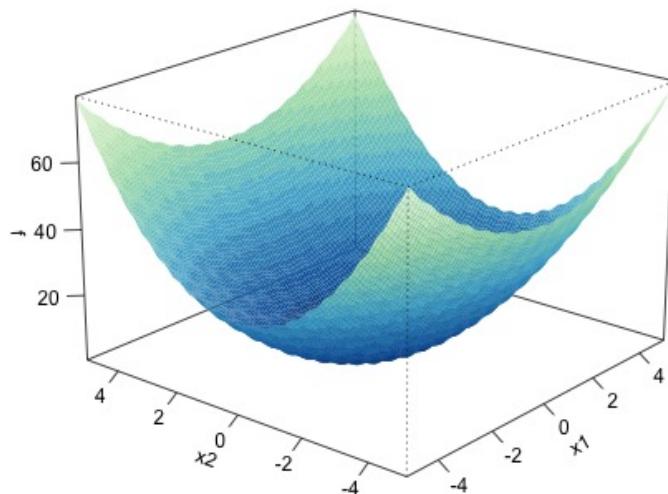
4 Funkcja Bohachevsky'ego

4.1 Wzór analityczny

$$f(x, y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7$$

Rysunek 35: Wzór analityczny funkcji Bochachevsky'ego

4.2 Wykres funkcji

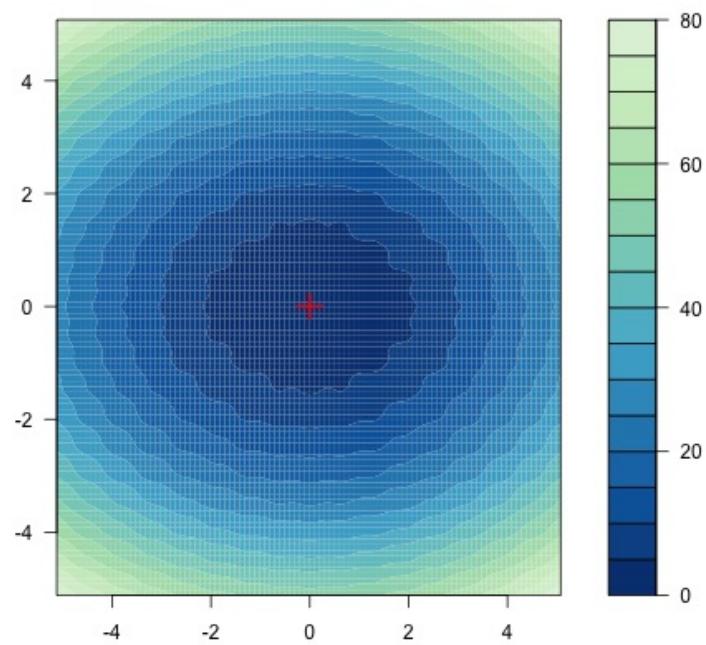


Rysunek 36: Wzór analityczny funkcji Bochachevsky'ego

4.3 Ekstremum globalne

$$f_j(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, 0), \text{ for all } j = 1, 2, 3$$

Rysunek 37: Minimum globalne funkcji Bochachevsky'ego

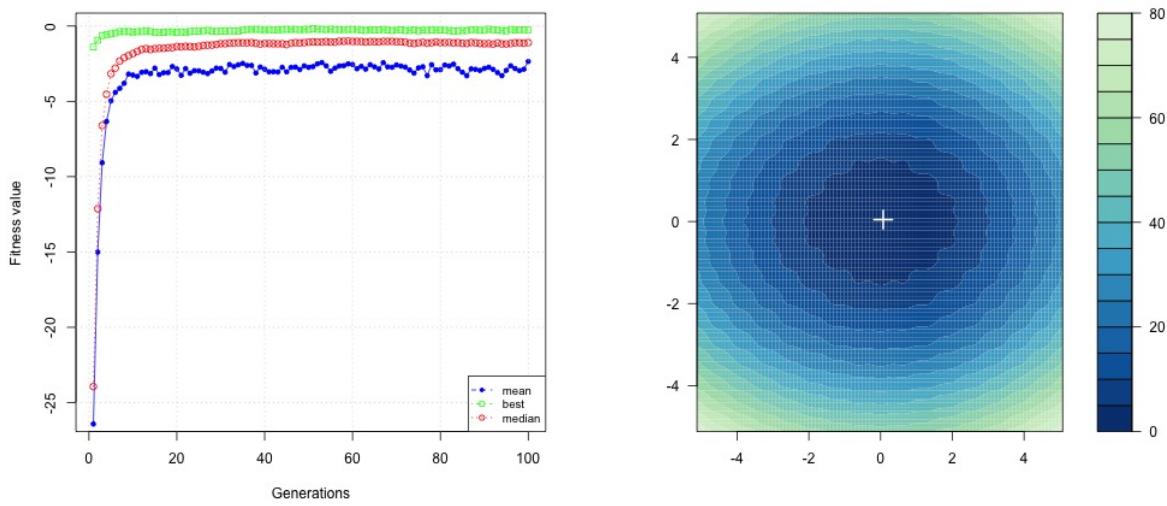


Rysunek 38: Minimum globalne funkcji Bochachevsky'ego

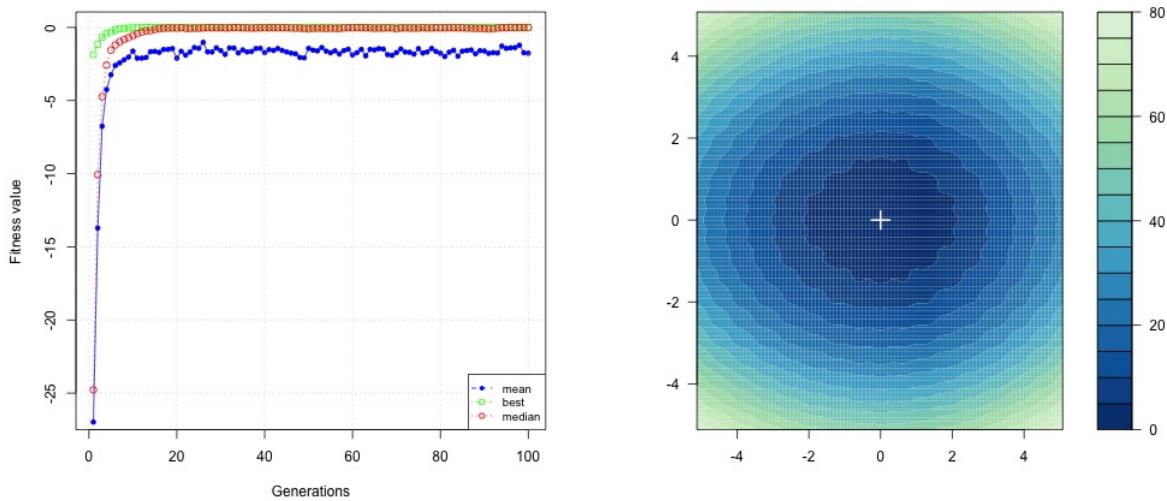
4.4 Optymalizacja

4.4.1 Modyfikacja parametru elitarności

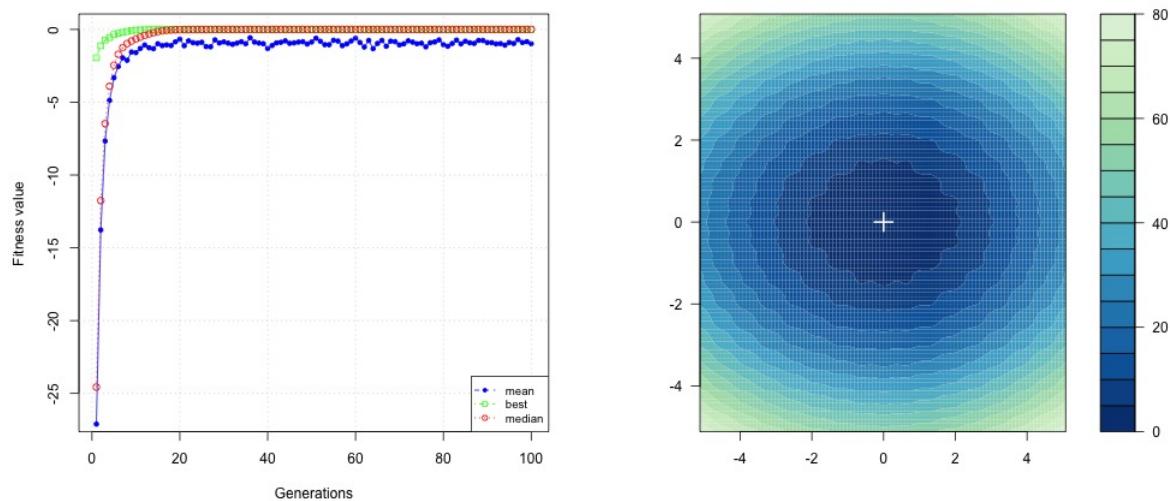
W przypadku zwiększenia wartości elitarności populacji średnia i mediana pozostają na podobnym poziomie. W przypadku większej elitarności algorytm szybciej znajduje optymalne rozwiązanie. Jeżeli elitarność populacji jest równa 1, to algorytm stochastycznie wlosował jeden zestaw rozwiązań, który z powodu braku ewolucji nie zbliżył się do rozwiązania optymalnego.



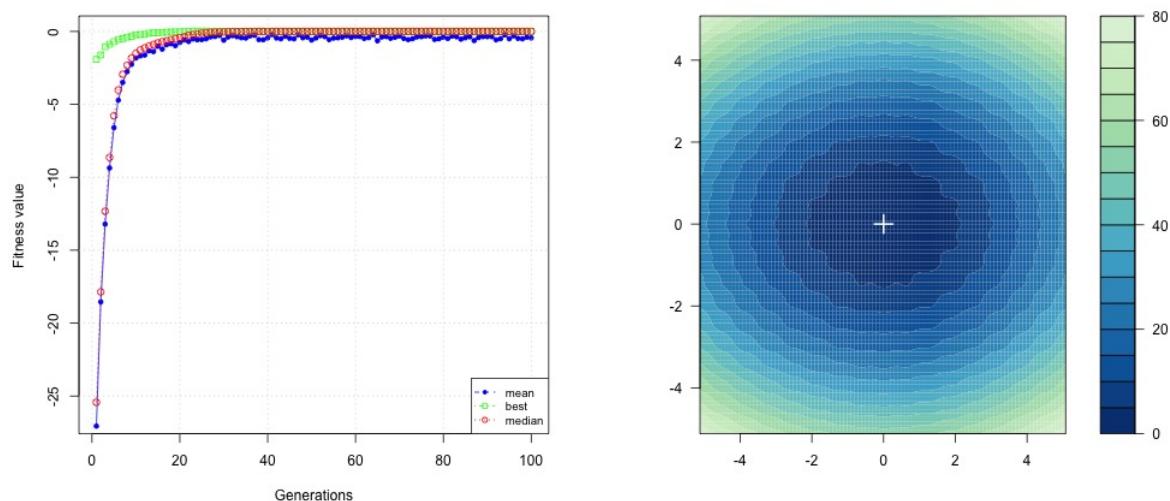
Rysunek 39: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e0



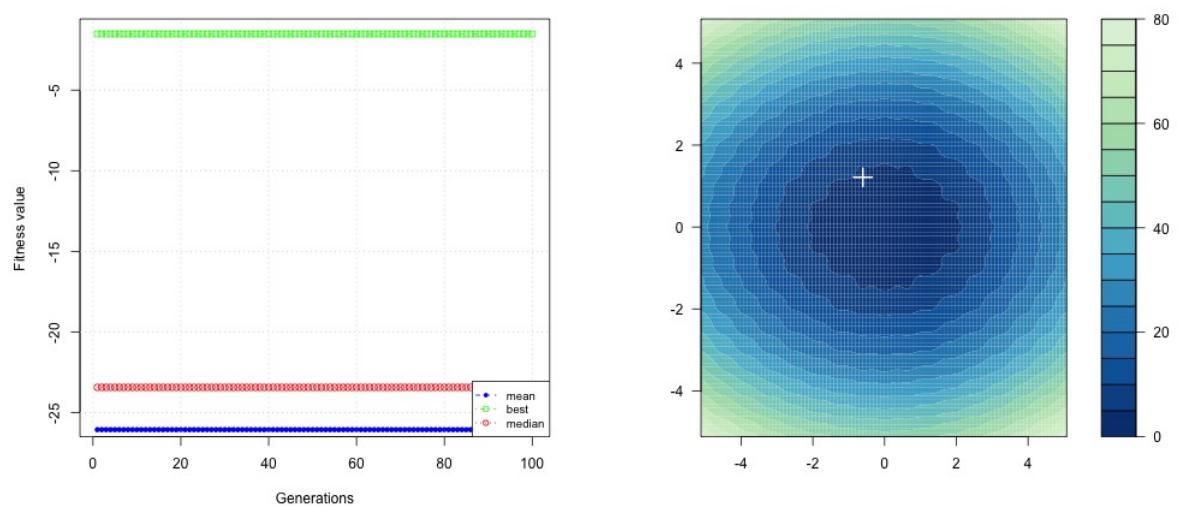
Rysunek 40: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e0.25



Rysunek 41: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e0.5



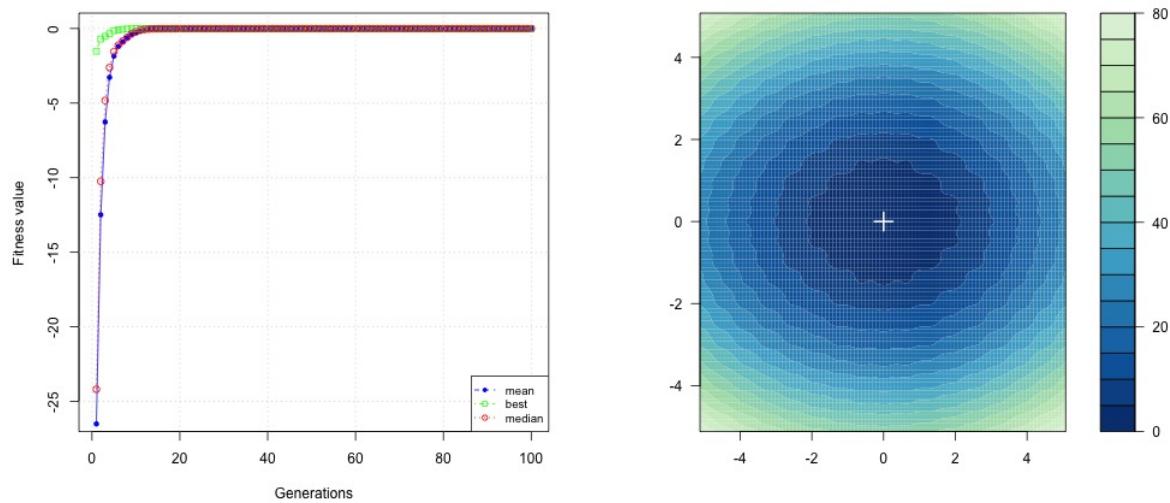
Rysunek 42: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e0.75



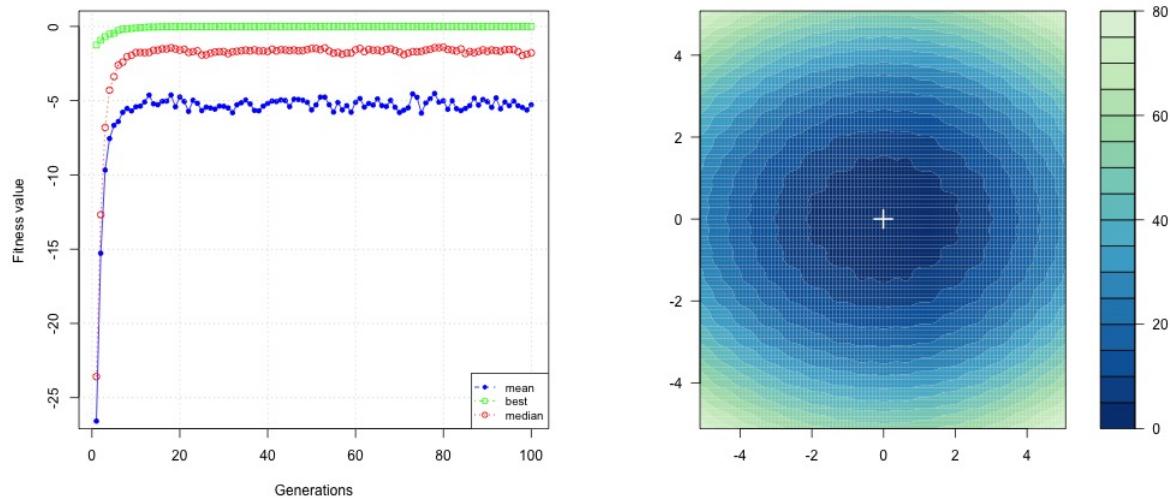
Rysunek 43: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e1

4.4.2 Modyfikacja parametru mutacji

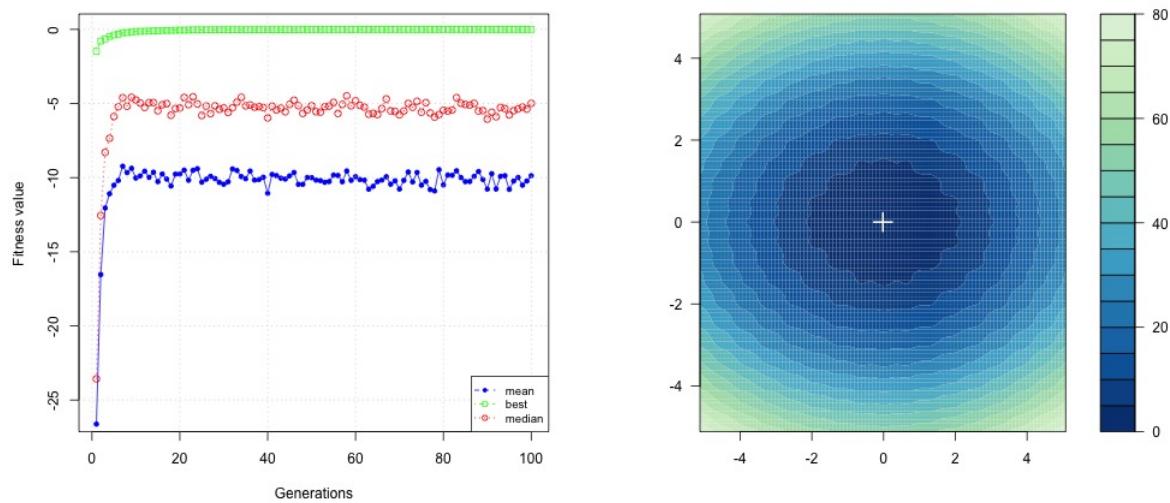
Zwiększenie parametru mutacji sprawia, że zarówno średnia jak i mediana zwiększają się wraz ze wzrostem parametru mutacji. Niezależnie od tego algorytm znajduje optymalne rozwiązanie stosunkowo szybko i po podobnej ilości pokoleń.



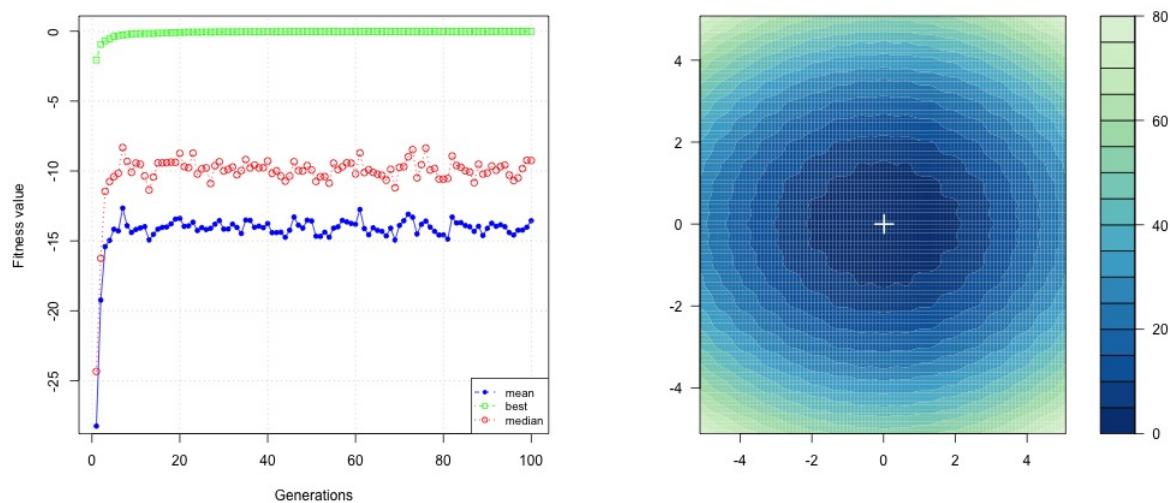
Rysunek 44: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0 e0.05



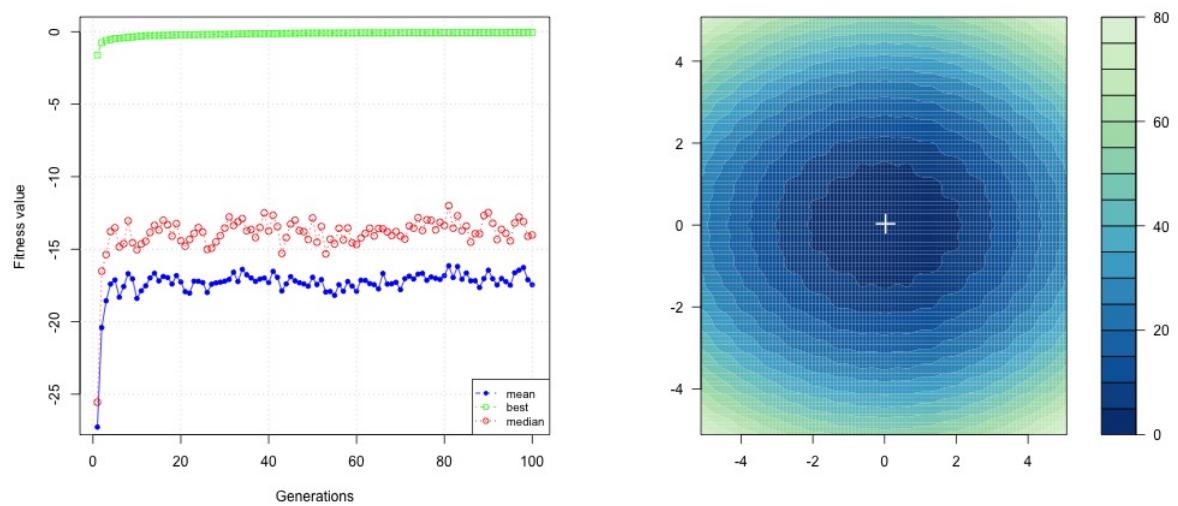
Rysunek 45: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.25 e0.05



Rysunek 46: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.5 e0.05



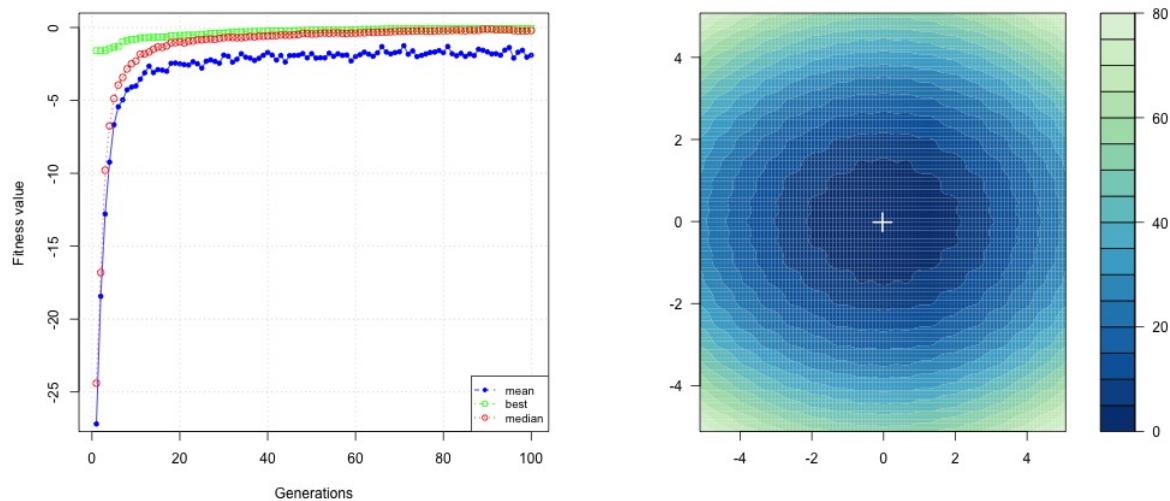
Rysunek 47: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.75 e0.05



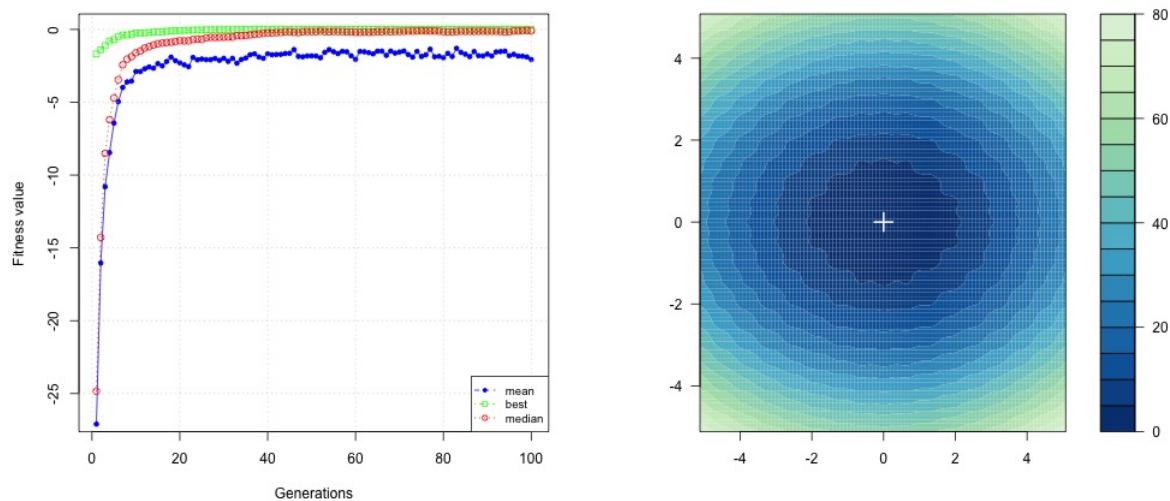
Rysunek 48: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m1 e0.05

4.4.3 Modyfikacja parametru krzyżowania

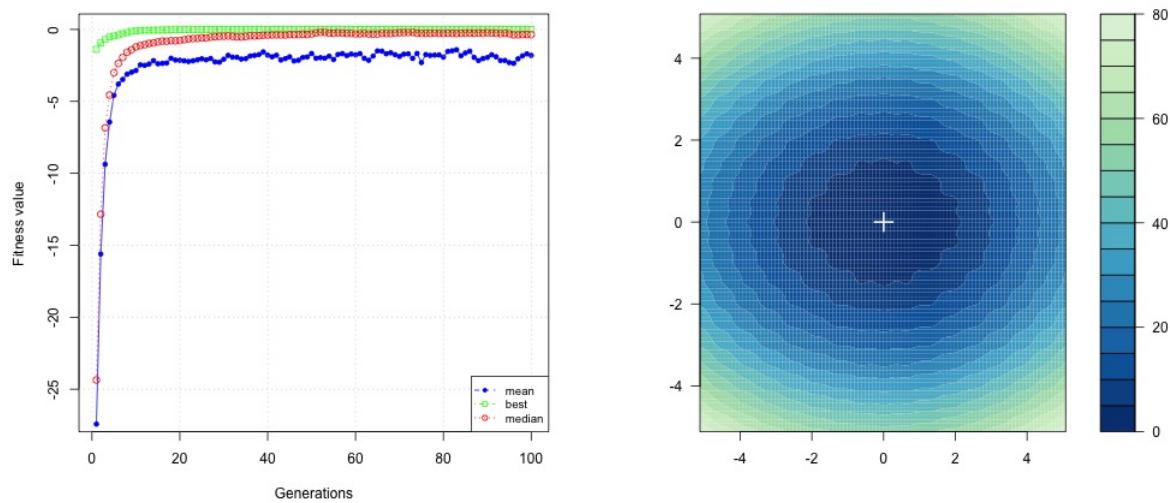
Modyfikacja parametru krzyżowania nie wpływa na rozwiązanie, ani na wartości średniej i mediany. W odróżnieniu od funkcji Schuberta, funkcja Bohachevsky'ego nie posiada wielu minimum lokalnych, a tylko jedno ekstremum, które okazuje się globalne.



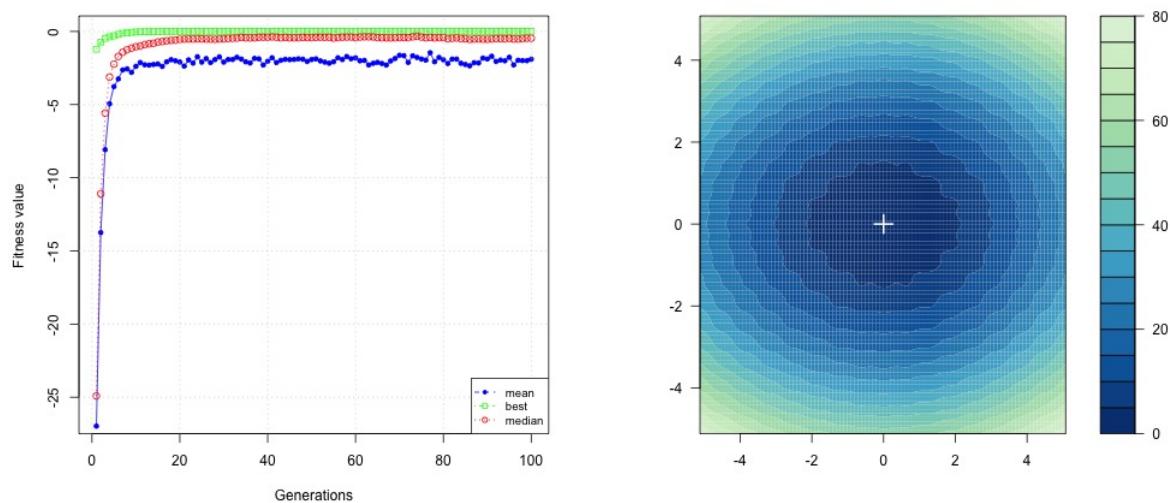
Rysunek 49: Test optymalizacji GA Bohachevsky1 p50 i100 c0 m0.1 e0.05



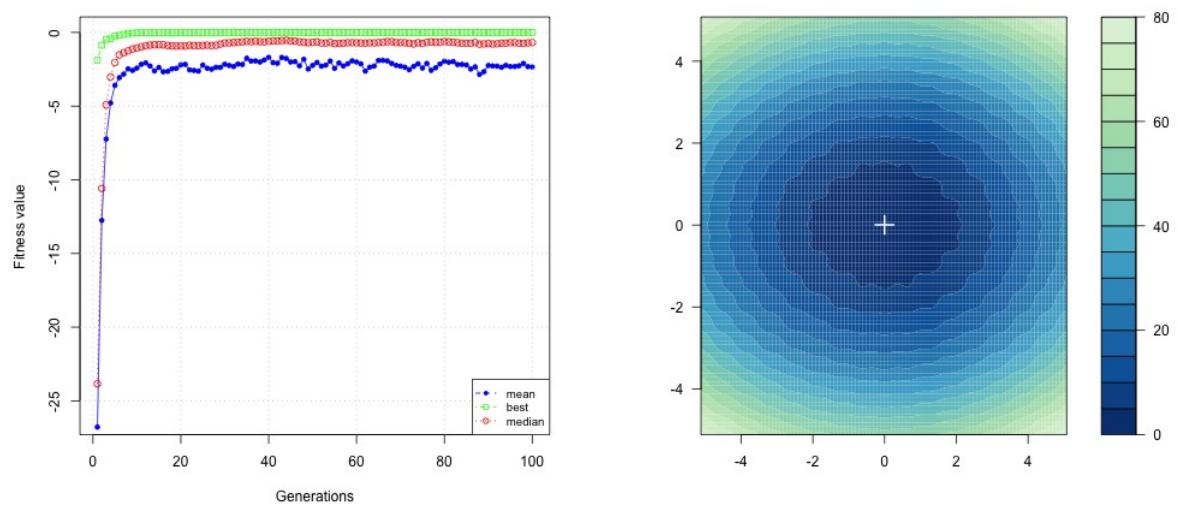
Rysunek 50: Test optymalizacji GA Bohachevsky1 p50 i100 c0.25 m0.1 e0.05



Rysunek 51: Test optymalizacji GA Bohachevsky1 p50 i100 c0.5 m0.1 e0.05



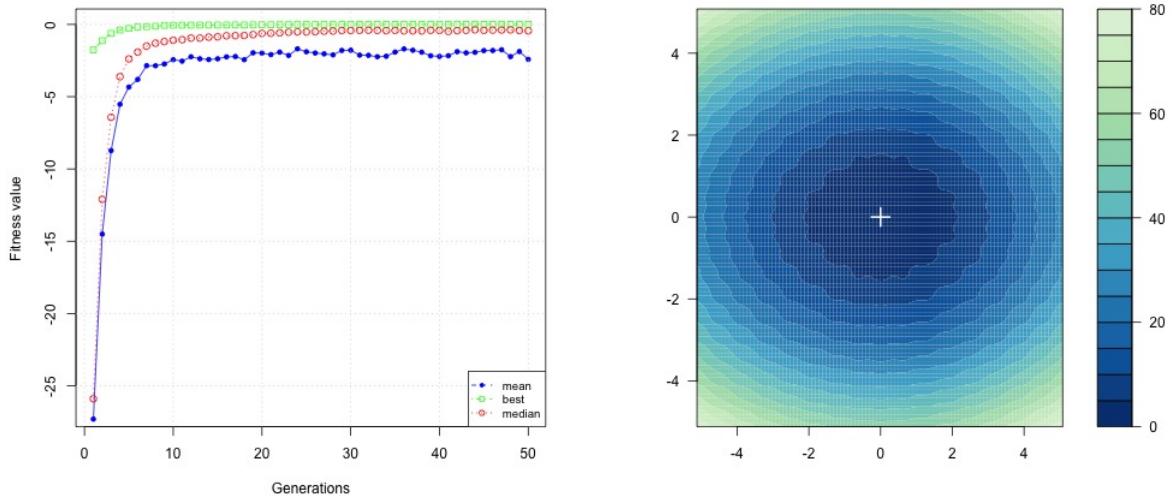
Rysunek 52: Test optymalizacji GA Bohachevsky1 p50 i100 c0.75 m0.1 e0.05



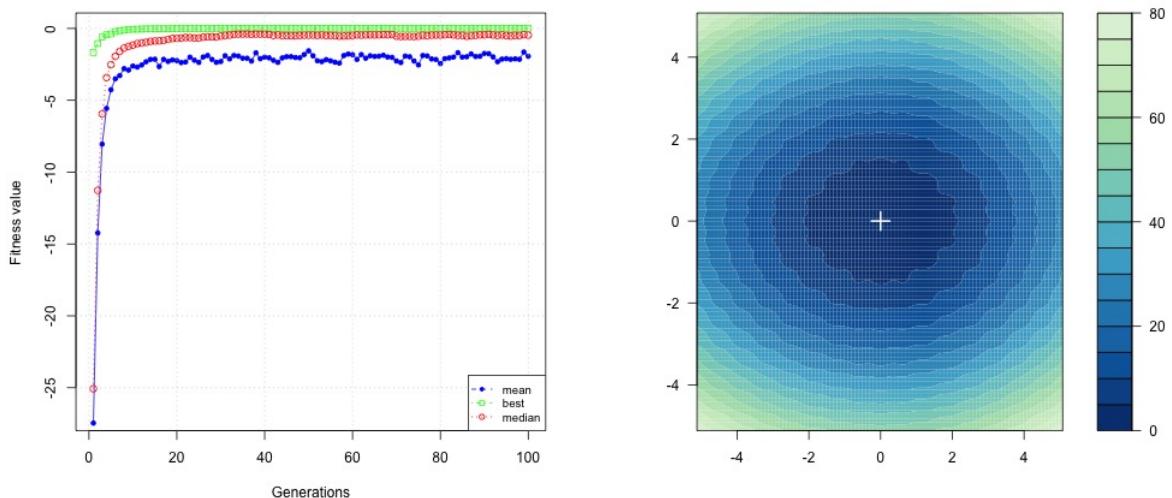
Rysunek 53: Test optymalizacji GA Bohachevsky1 p50 i100 c1 m0.1 e0.05

4.4.4 Modyfikacja parametru liczby iteracji

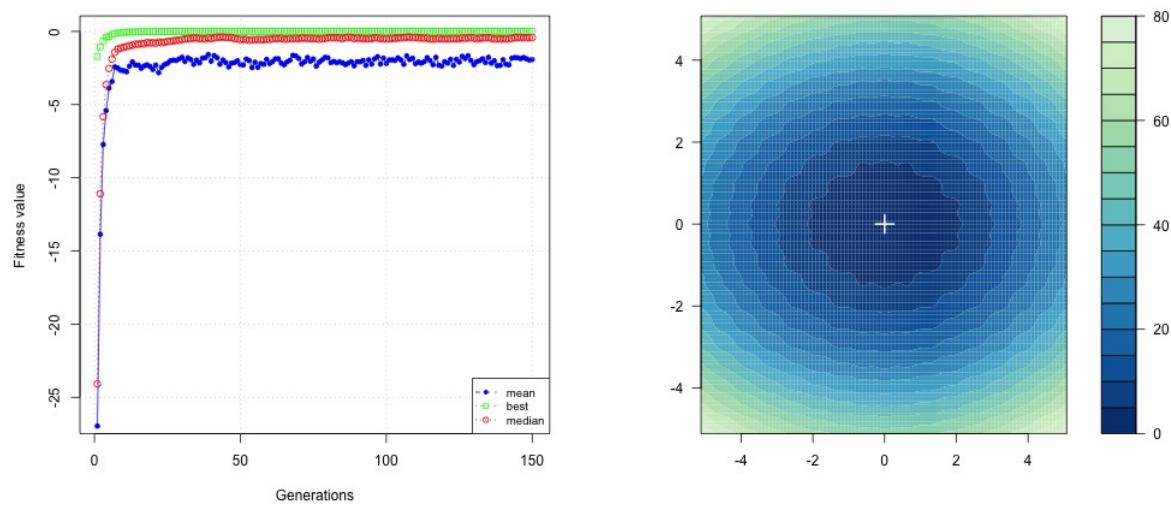
Modyfikacja parametru liczby iteracji nie wpływa na rozwiązańie, ani na wartości średniej i mediany. Sytuacja jest analogiczna do przypadku modyfikacji parametru krzyżowania.



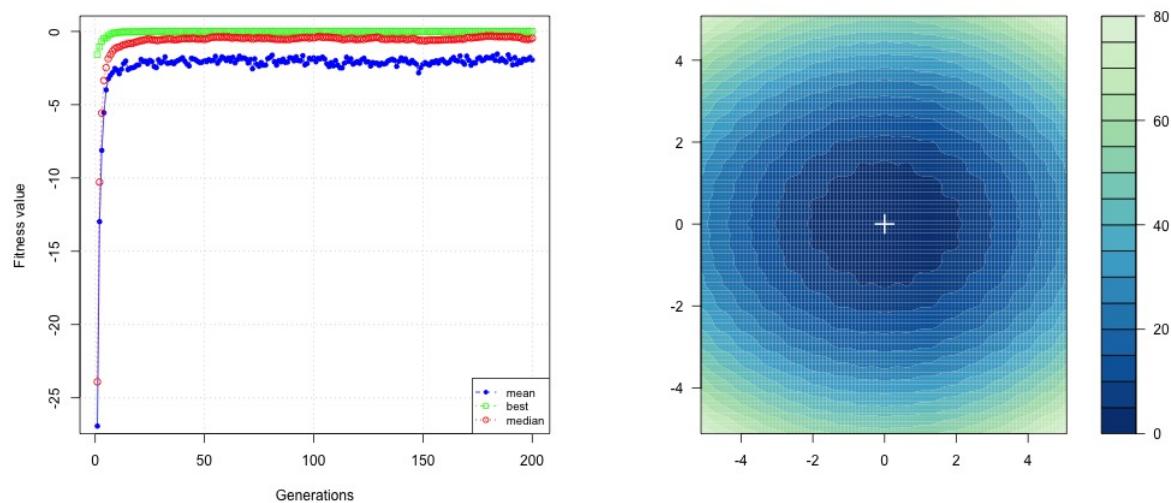
Rysunek 54: Test optymalizacji GA Bohachevsky1 p50 i50 c0.8 m0.1 e0.05



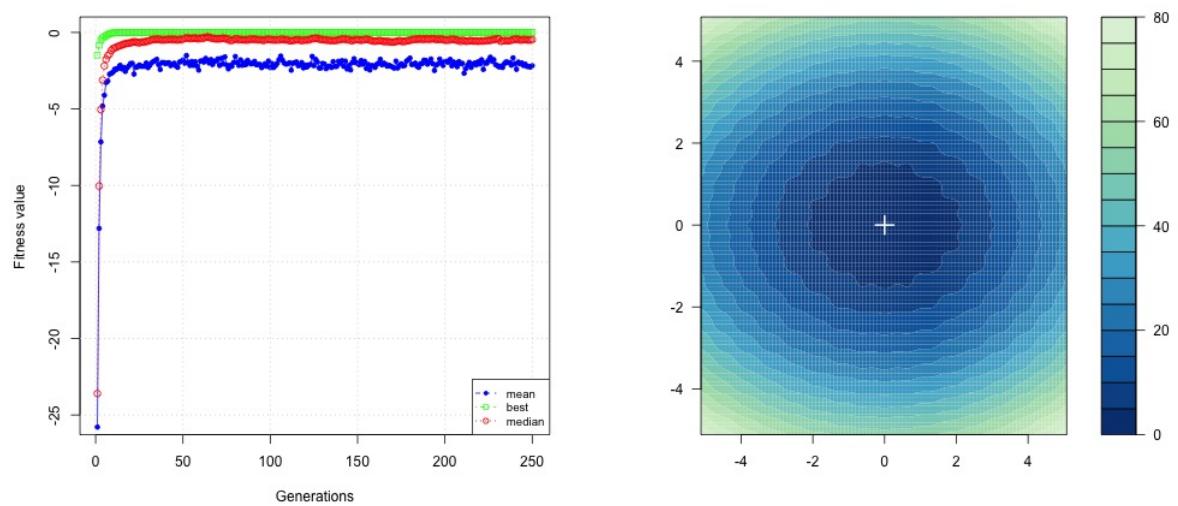
Rysunek 55: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e0.05



Rysunek 56: Test optymalizacji GA Bohachevsky1 p50 i150 c0.8 m0.1 e0.05



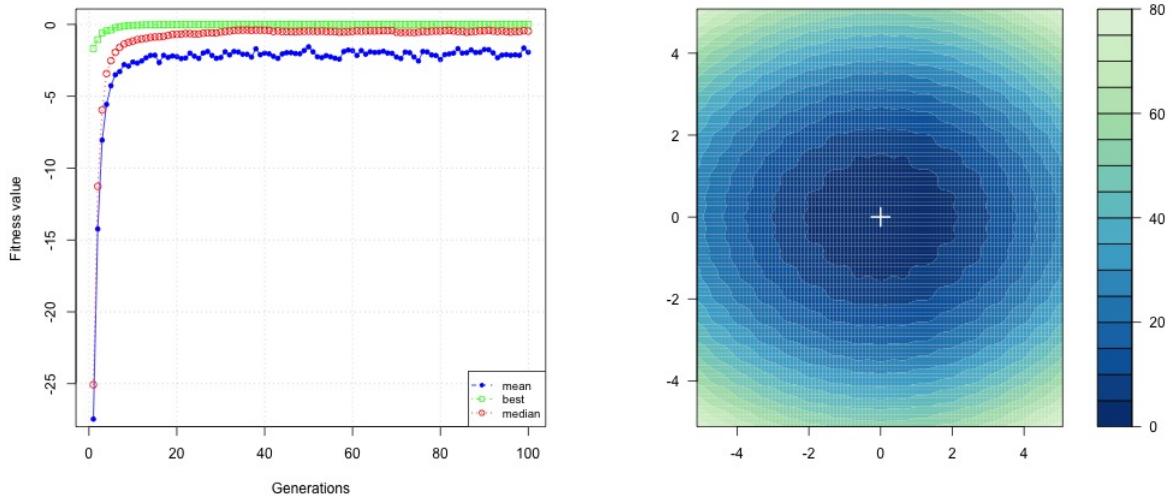
Rysunek 57: Test optymalizacji GA Bohachevsky1 p50 i200 c0.8 m0.1 e0.05



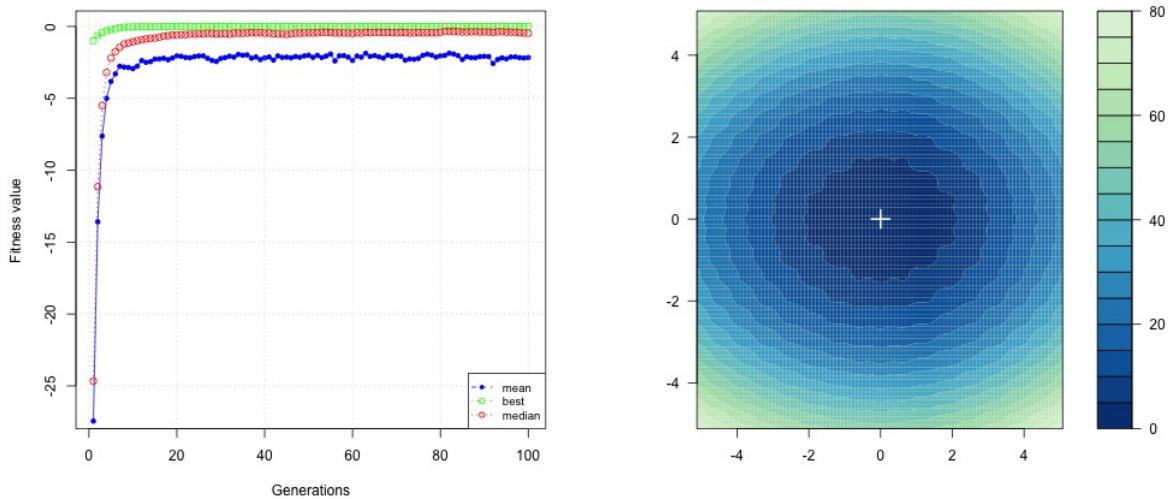
Rysunek 58: Test optymalizacji GA Bohachevsky1 p50 i250 c0.8 m0.1 e0.05

4.4.5 Modyfikacja parametru rozmiaru populacji

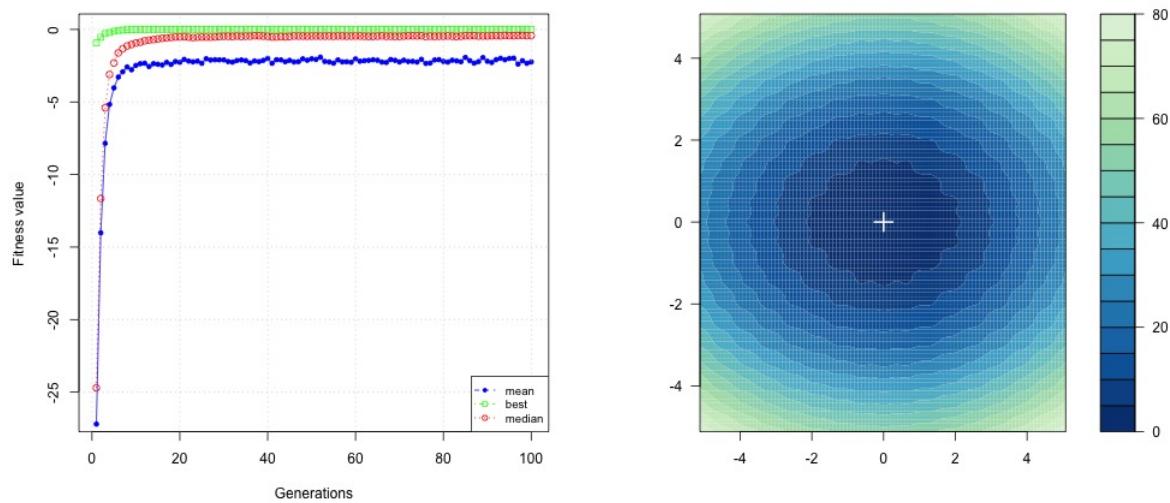
Modyfikacja parametru rozmiaru populacji nie wpływa na rozwiązanie, ani na wartości średniej i mediany. Sytuacja jest analogiczna do przypadku modyfikacji parametru krzyżowania.



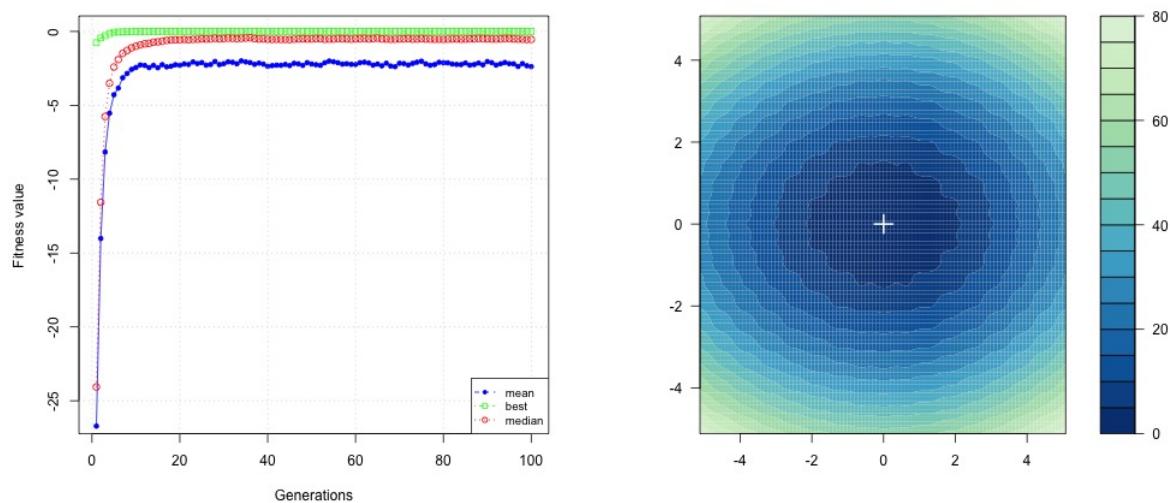
Rysunek 59: Test optymalizacji GA Bohachevsky1 p50 i100 c0.8 m0.1 e0.05



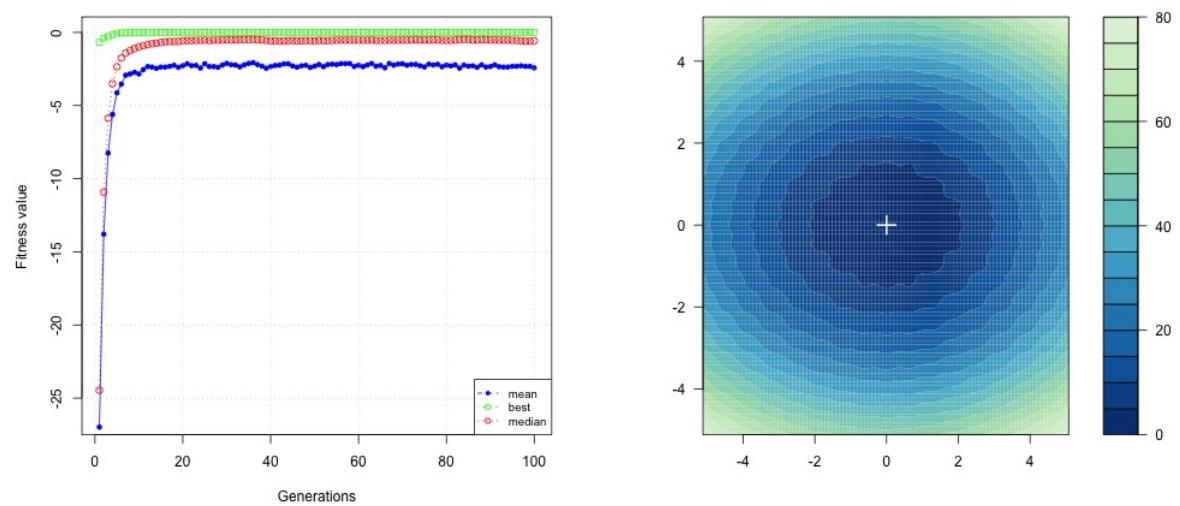
Rysunek 60: Test optymalizacji GA Bohachevsky1 p100 i100 c0.8 m0.1 e0.05



Rysunek 61: Test optymalizacji GA Bohachevsky1 p150 i100 c0.8 m0.1 e0.05



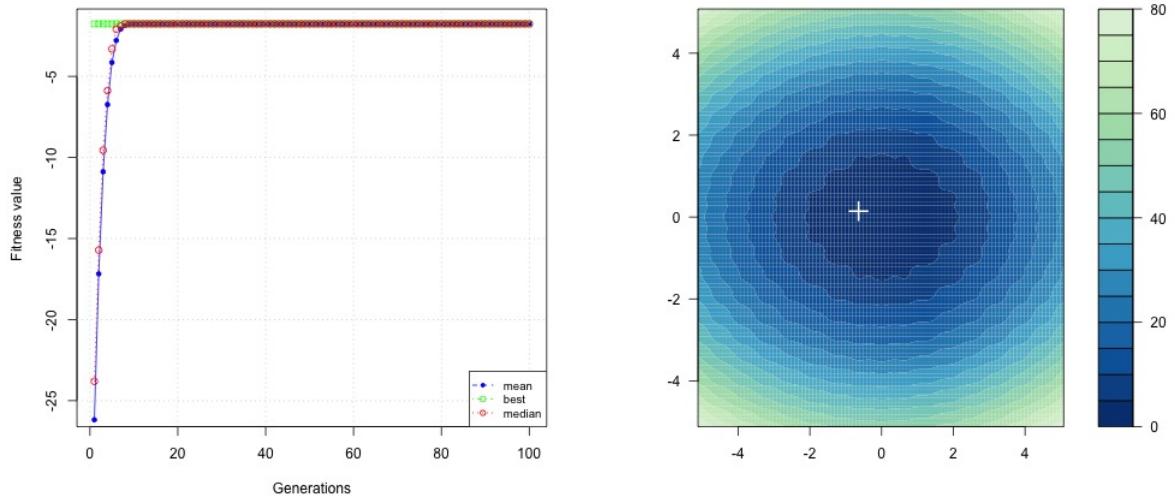
Rysunek 62: Test optymalizacji GA Bohachevsky1 p200 i100 c0.8 m0.1 e0.05



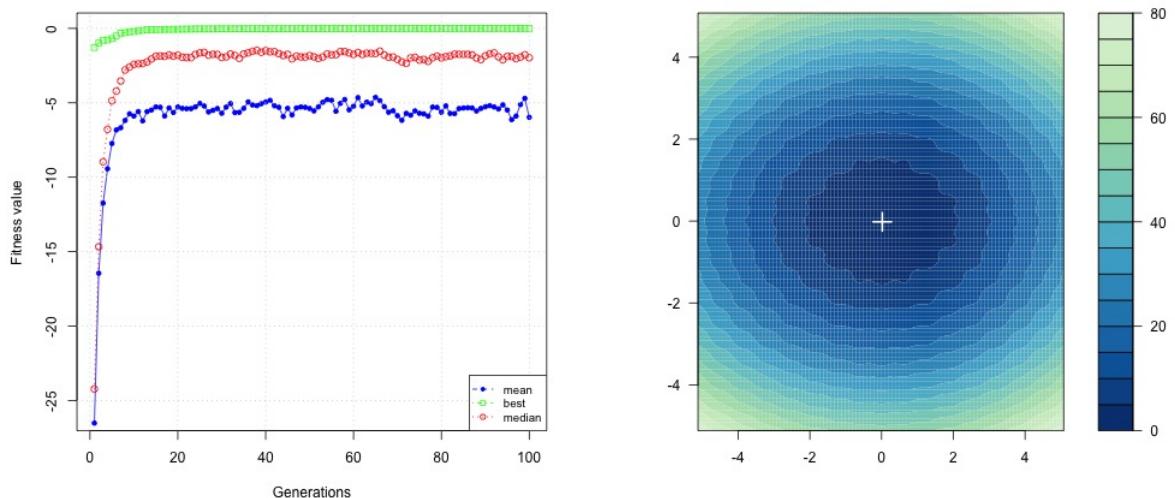
Rysunek 63: Test optymalizacji GA Bohachevsky1 p250 i100 c0.8 m0.1 e0.05

4.5 Jednoczesna modyfikacja parametru krzyżowania i mutacji

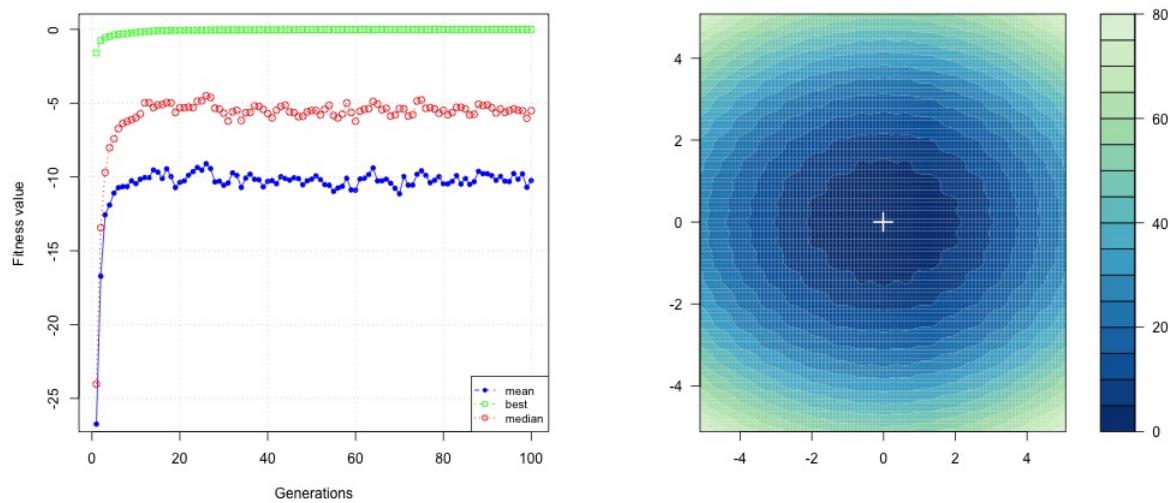
W przypadku jednocięsnej modyfikacji parametru krzyżowania i mutacji krzywe rozwiązań mają wartości podobne. Wraz ze wzrostem obydwu parametrów wartości średniej i mediany są gorsze.



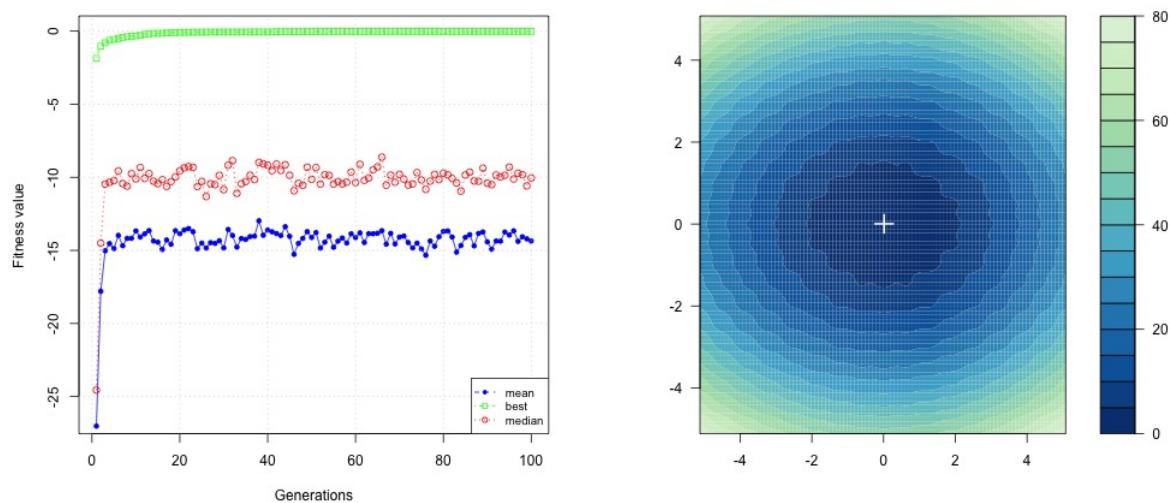
Rysunek 64: Test optymalizacji GA Bohachevsky1 p050 i100 c0.0 m0.0 e0.05



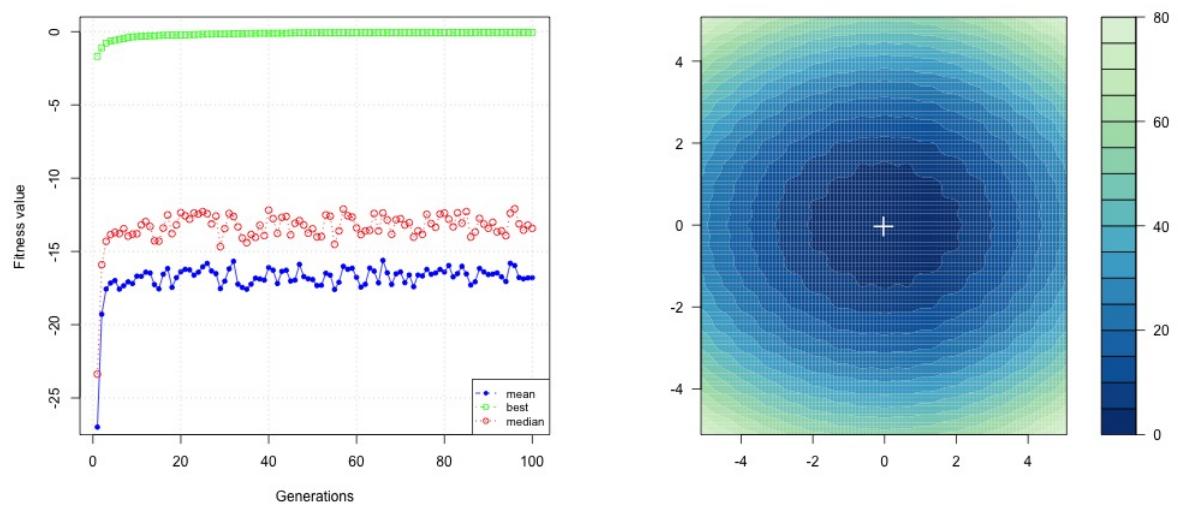
Rysunek 65: Test optymalizacji GA Bohachevsky1 p050 i100 c0.25 m0.25 e0.05



Rysunek 66: Test optymalizacji GA Bohachevsky1 p050 i100 c0.50 m0.50 e0.05



Rysunek 67: Test optymalizacji GA Bohachevsky1 p050 i100 c0.75 m0.75 e0.05



Rysunek 68: Test optymalizacji GA Bohachevsky1 p050 i100 c1.00 m1.00 e0.05

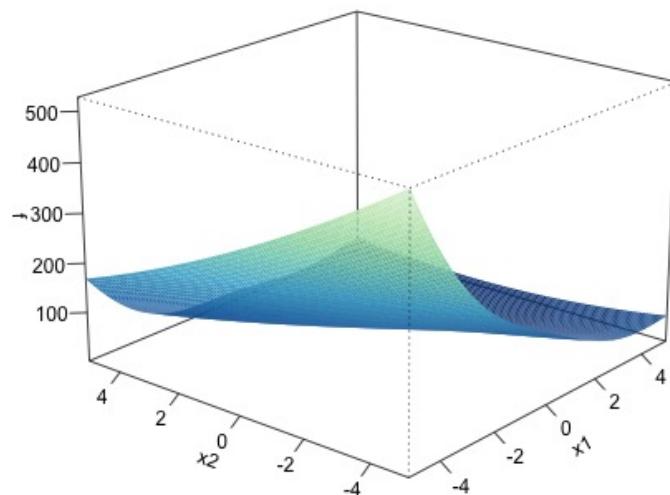
5 Funkcja Branina

5.1 Wzór analityczny

$$f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t)\cos(x_1) + s$$

Rysunek 69: Wzór analityczny funkcji Branina

5.2 Wykres w ustalonym przedziale zmiennych

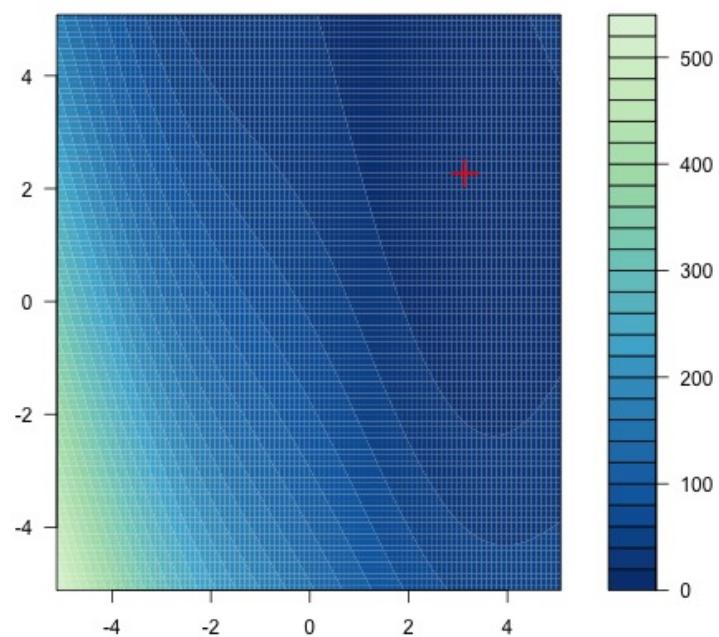


Rysunek 70: Wzór analityczny funkcji Branina

5.3 Ekstremum globalne

$$f(\mathbf{x}^*) = 0.397887, \text{ at } \mathbf{x}^* = (-\pi, 12.275), (\pi, 2.275) \text{ and } (9.42478, 2.475)$$

Rysunek 71: Minimum globalne funkcji Branina

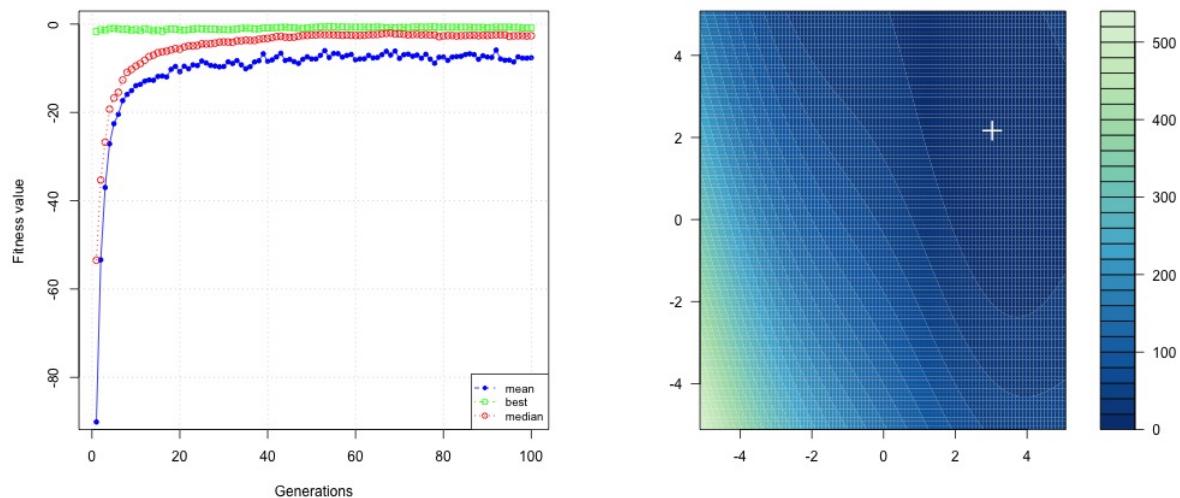


Rysunek 72: Minimum globalne funkcji Branina

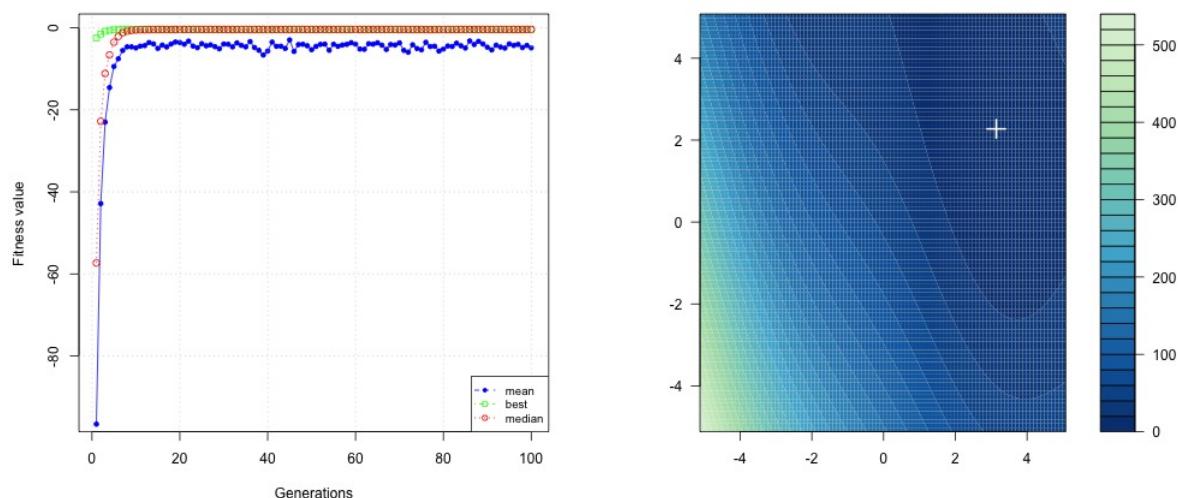
5.4 Optymalizacja

5.4.1 Modyfikacja parametru elitarności

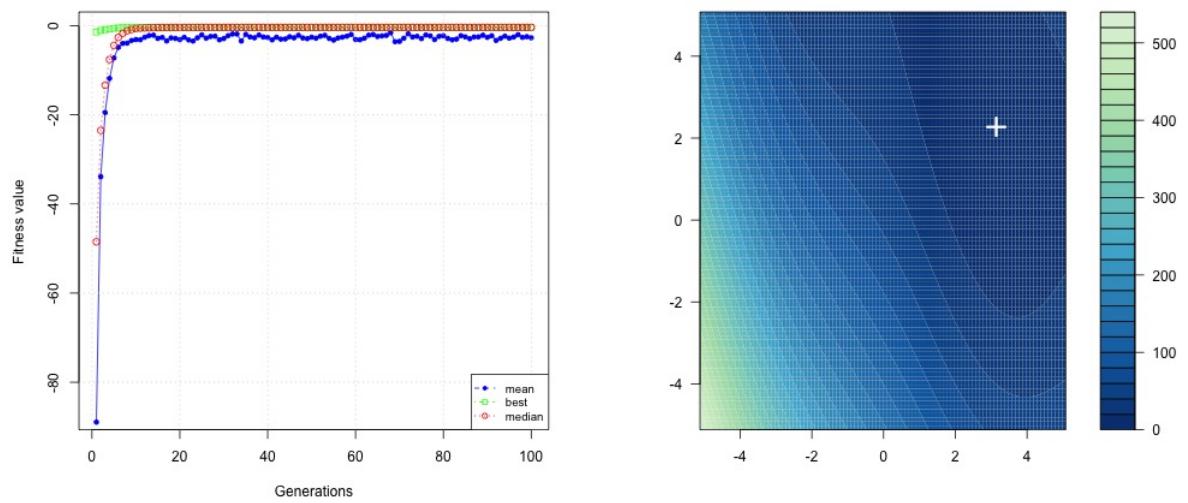
W przypadku braku populacji elitarnej algorytm potrzebował więcej czasu aby odnaleźć poprawny wynik, wskazuje na to wykres mediany. W przypadku wartości 0.25, 0.5 oraz 0.75 algorytm szybciej odnajdywał poprawne rozwiązanie. W momencie gdy cała populacja była oznaczana jako elitarna algorytm nie znajdował poprawnego rozwiązania, nie występowała również zmiana między kolejnymi generacjami.



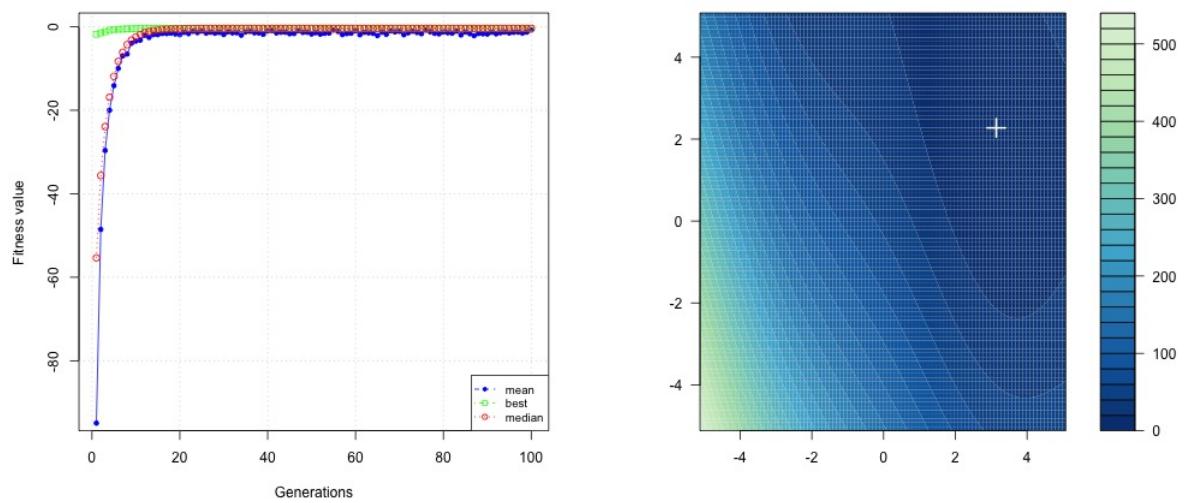
Rysunek 73: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e0



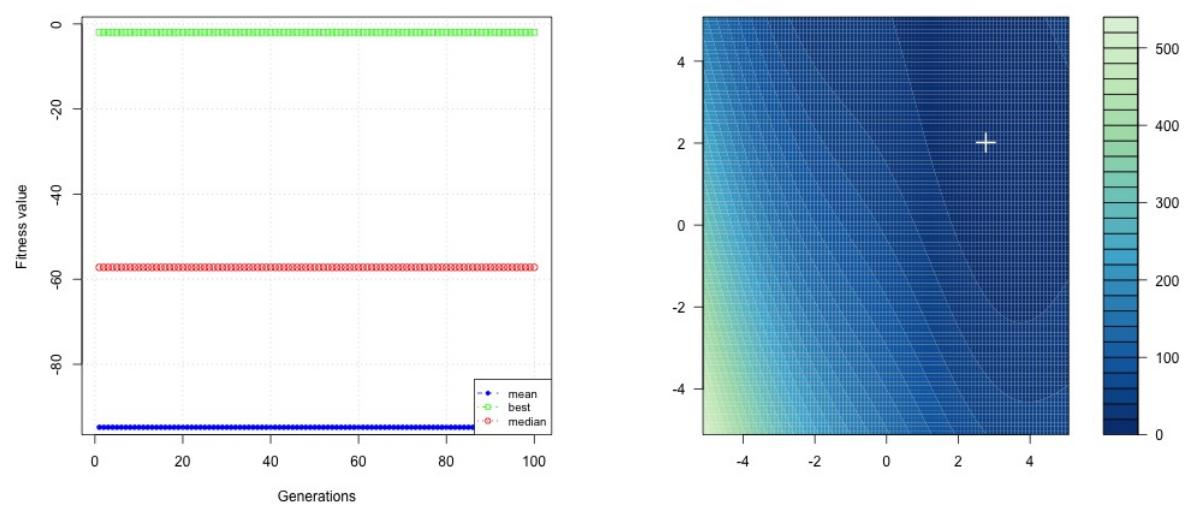
Rysunek 74: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e0.25



Rysunek 75: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e0.5



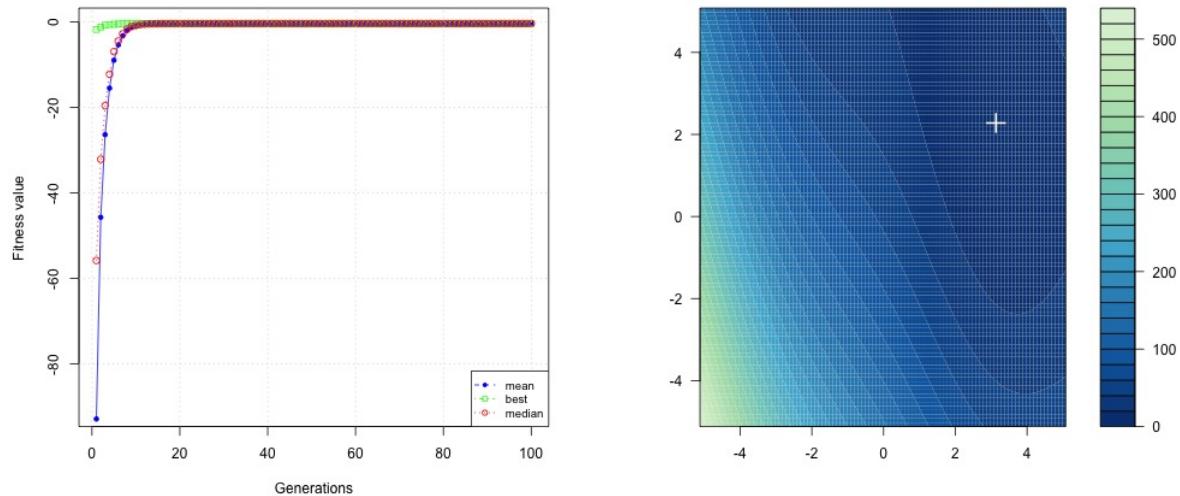
Rysunek 76: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e0.75



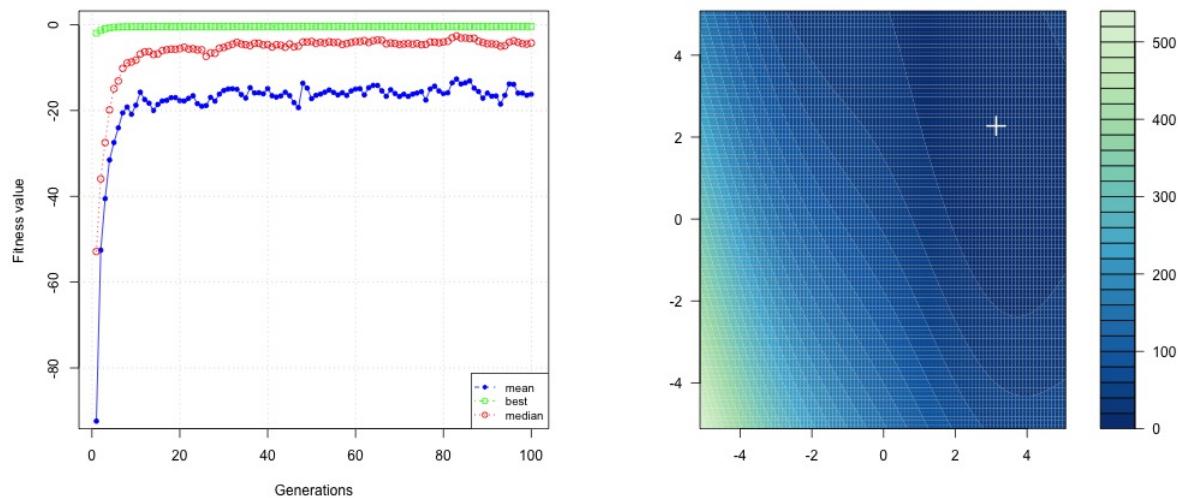
Rysunek 77: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e1

5.4.2 Modyfikacja parametru mutacji

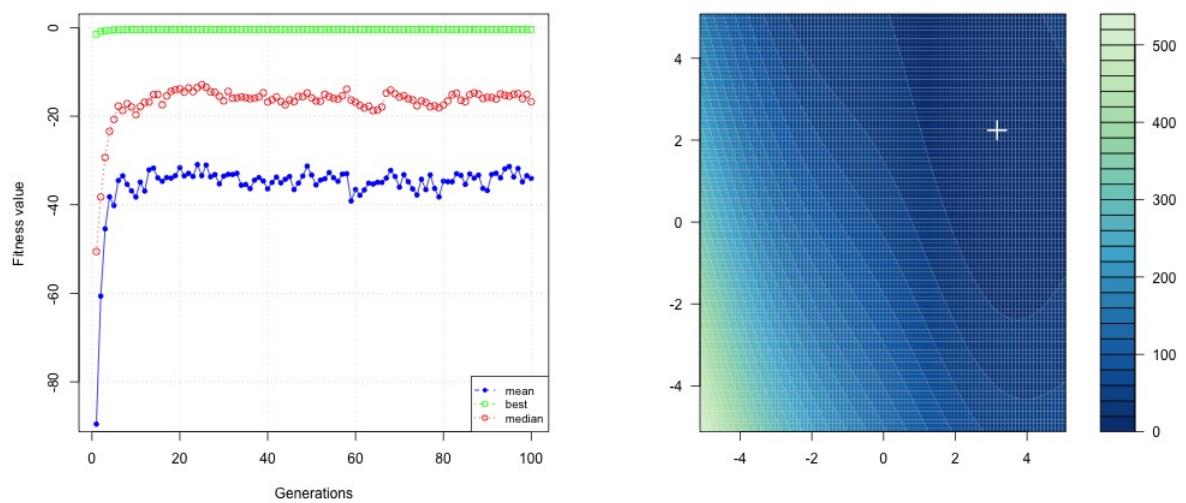
Algorytm bez wzgledu na zmianę parametru mutacji zawsze odnajdował poprawne rozwiązanie, im większa wartość parametru tym wykres mediany oraz średniej wskazywał na coraz większe odchylenia i niższe wyniki.



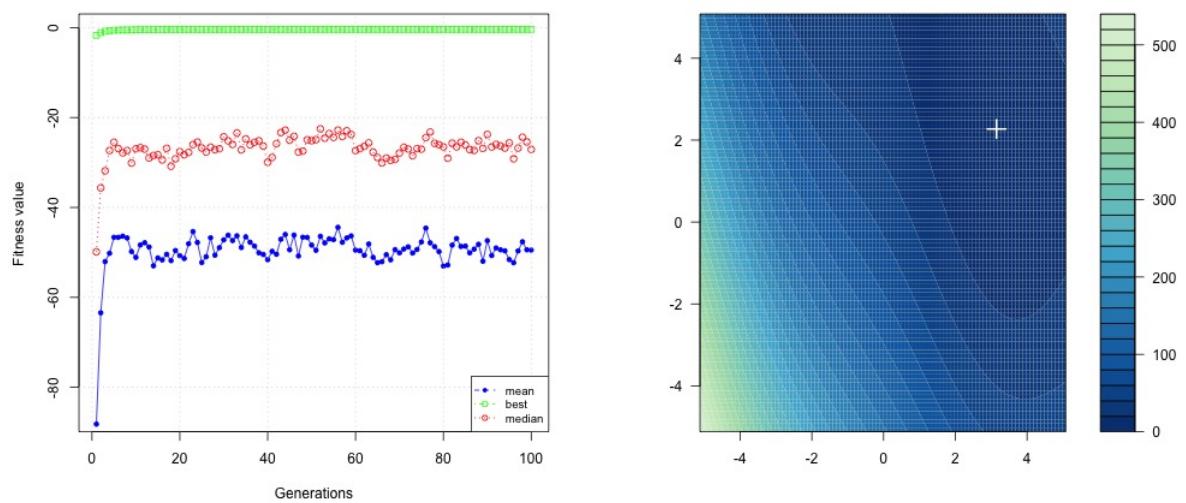
Rysunek 78: Test optymalizacji GA Branin p50 i100 c0.8 m0 e0.05



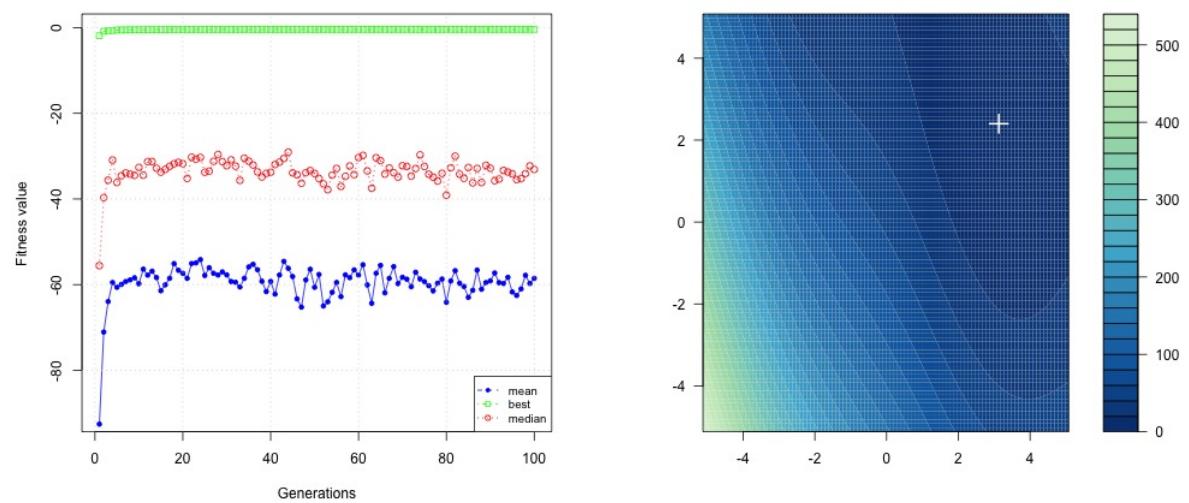
Rysunek 79: Test optymalizacji GA Branin p50 i100 c0.8 m0.25 e0.05



Rysunek 80: Test optymalizacji GA Branin p50 i100 c0.8 m0.5 e0.05



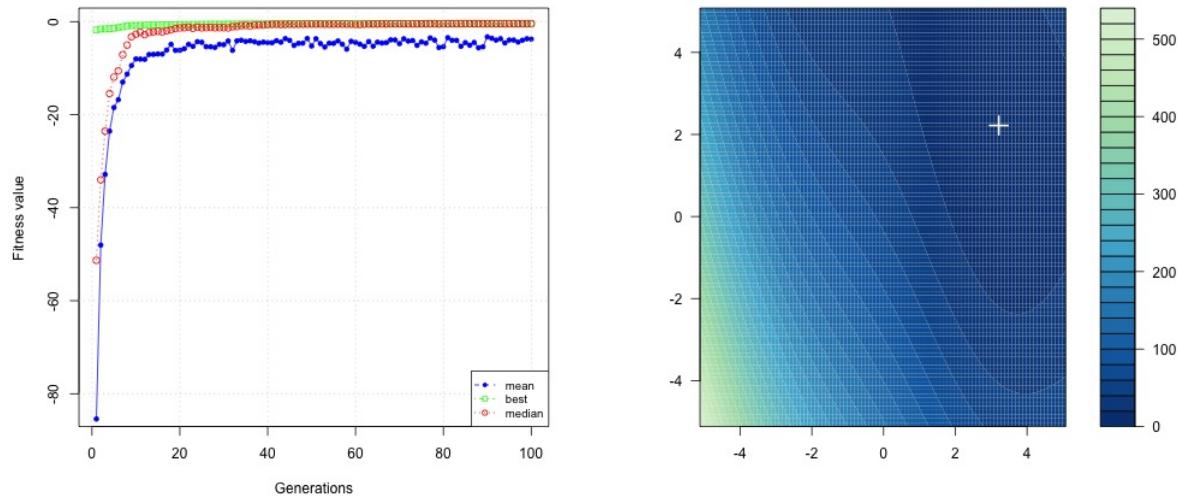
Rysunek 81: Test optymalizacji GA Branin p50 i100 c0.8 m0.75 e0.05



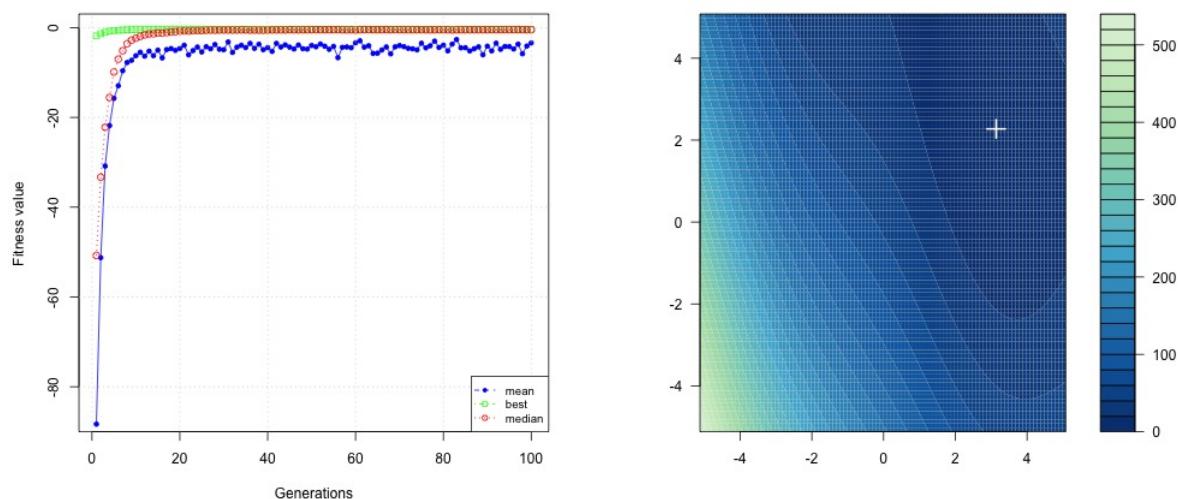
Rysunek 82: Test optymalizacji GA Branin p50 i100 c0.8 m1 e0.05

5.4.3 Modyfikacja parametru krzyżowania

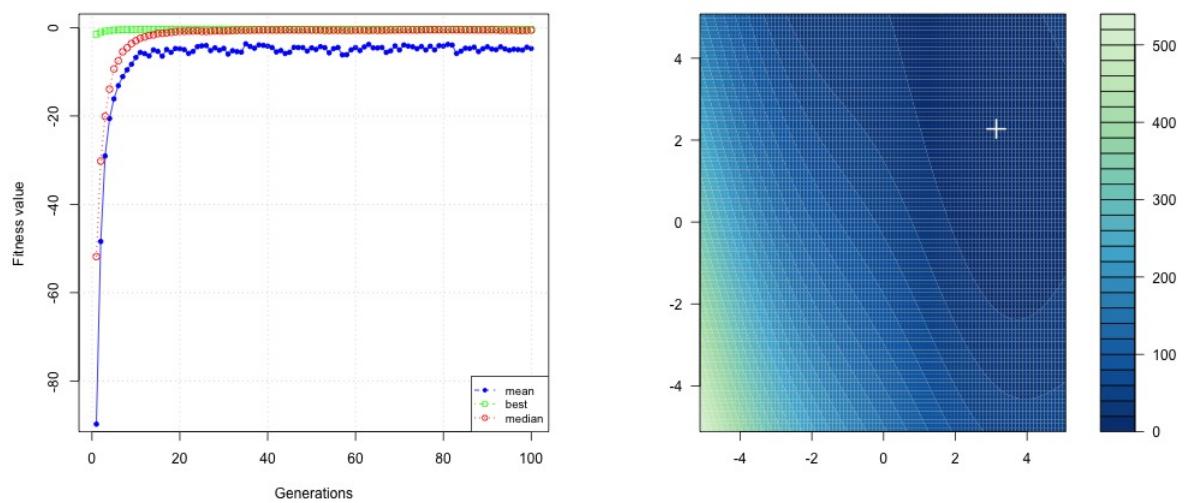
W przypadku braku krzyżowania funkcja potrzebuje większej ilości iteracji aby znaleźć poprawne rozwiązanie. Wyższy wskaźnik ma w tym przypadku pozytywny wpływ na szybkość odnajdywanego rozwiązania.



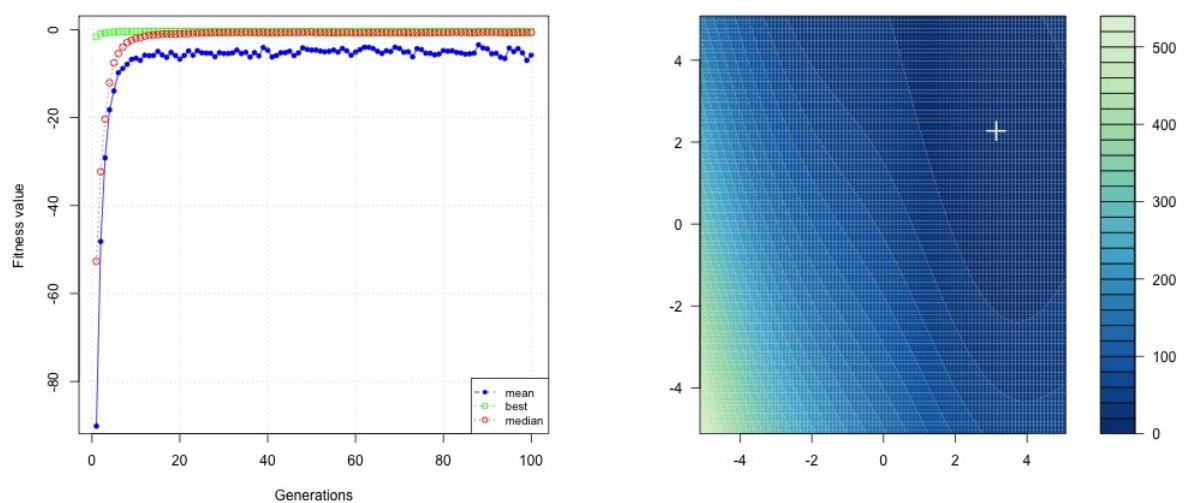
Rysunek 83: Test optymalizacji GA Branin p50 i100 c0 m0.1 e0.05



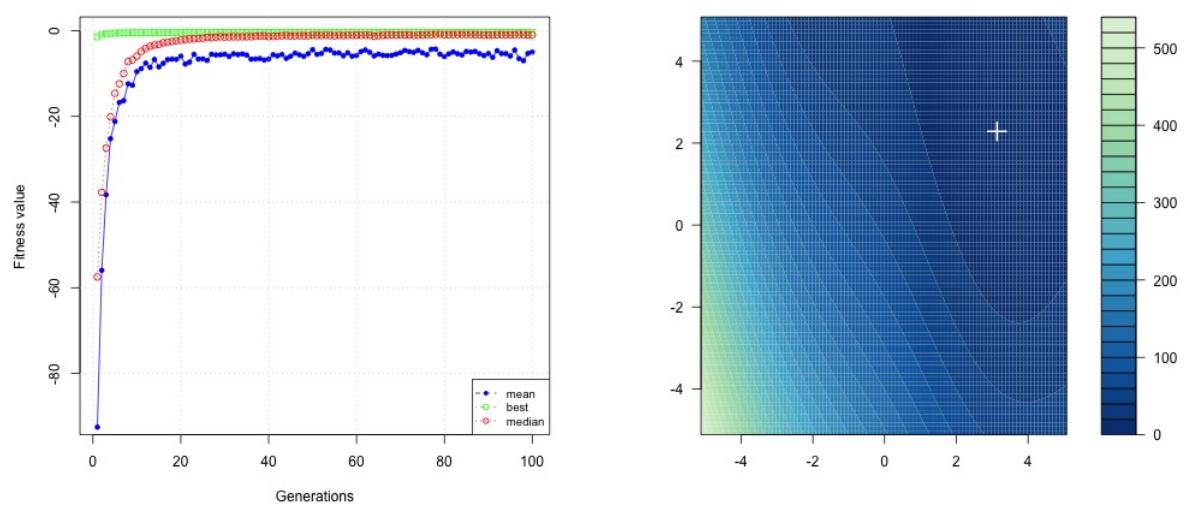
Rysunek 84: Test optymalizacji GA Branin p50 i100 c0.25 m0.1 e0.05



Rysunek 85: Test optymalizacji GA Branin p50 i100 c0.5 m0.1 e0.05



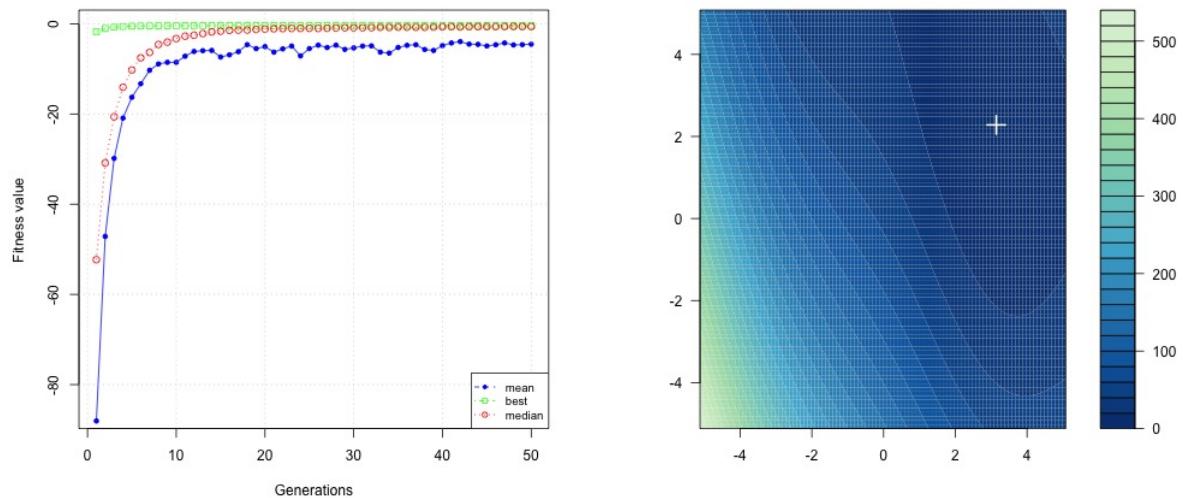
Rysunek 86: Test optymalizacji GA Branin p50 i100 c0.75 m0.1 e0.05



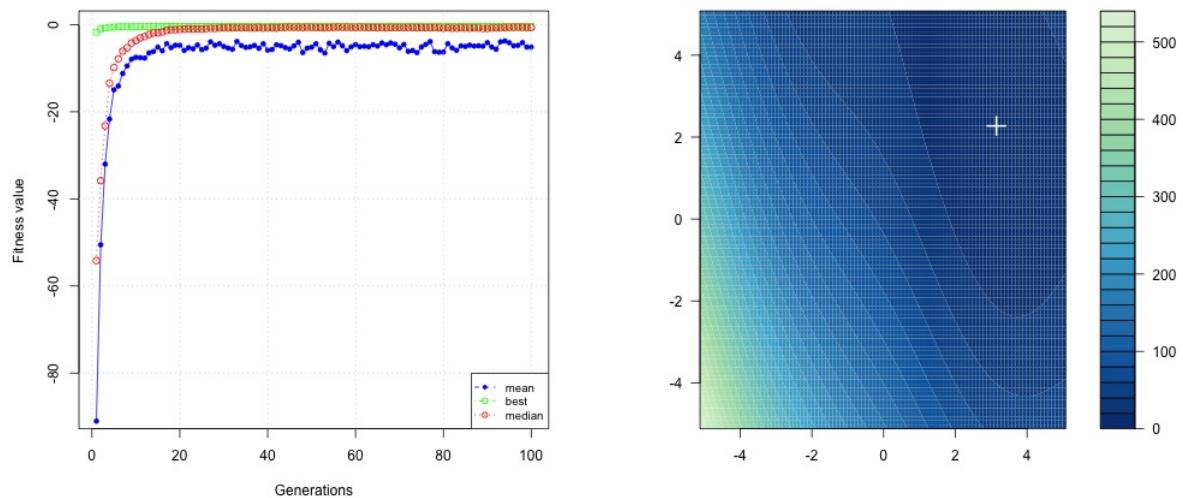
Rysunek 87: Test optymalizacji GA Branin p50 i100 c1 m0.1 e0.05

5.4.4 Modyfikacja parametru liczby iteracji

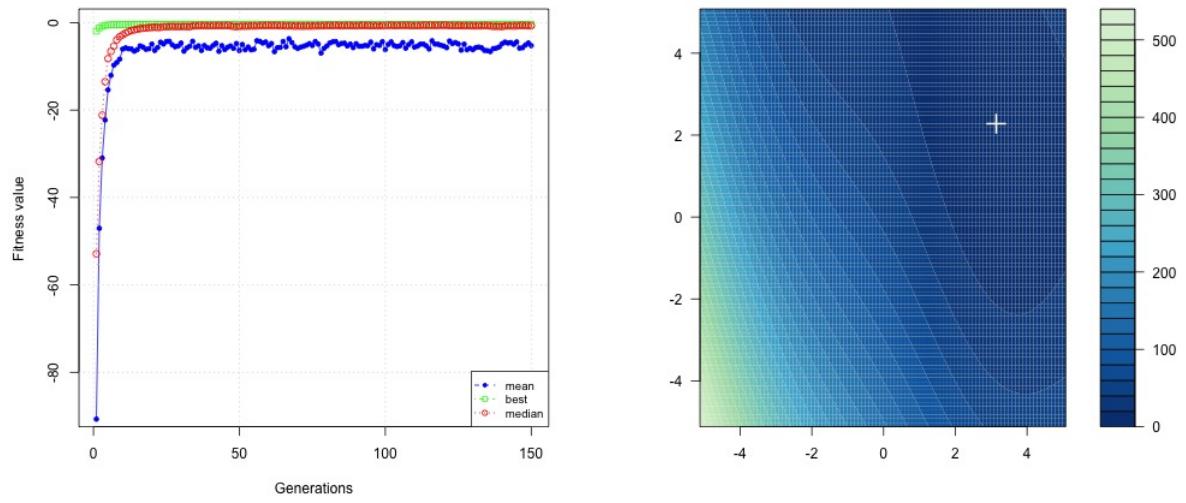
Zmiana liczby iteracji w przypadku funkcji Branin nie wpływa na jakość rozwiązania. Nie występują żadne anomalie między zmianami parametru. Funkcja w każdym przypadku zachowuje się identycznie.



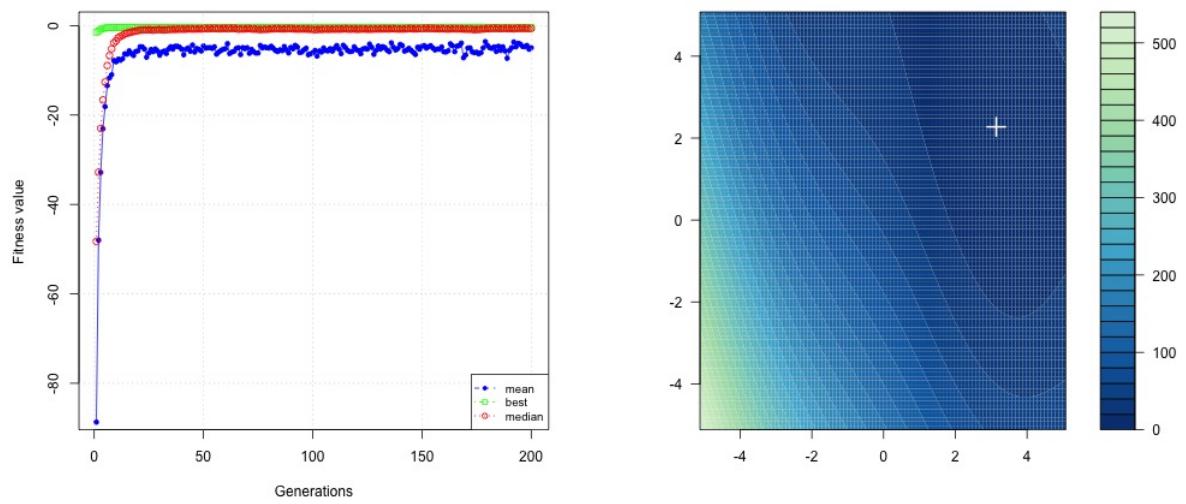
Rysunek 88: Test optymalizacji GA Branin p50 i50 c0.8 m0.1 e0.05



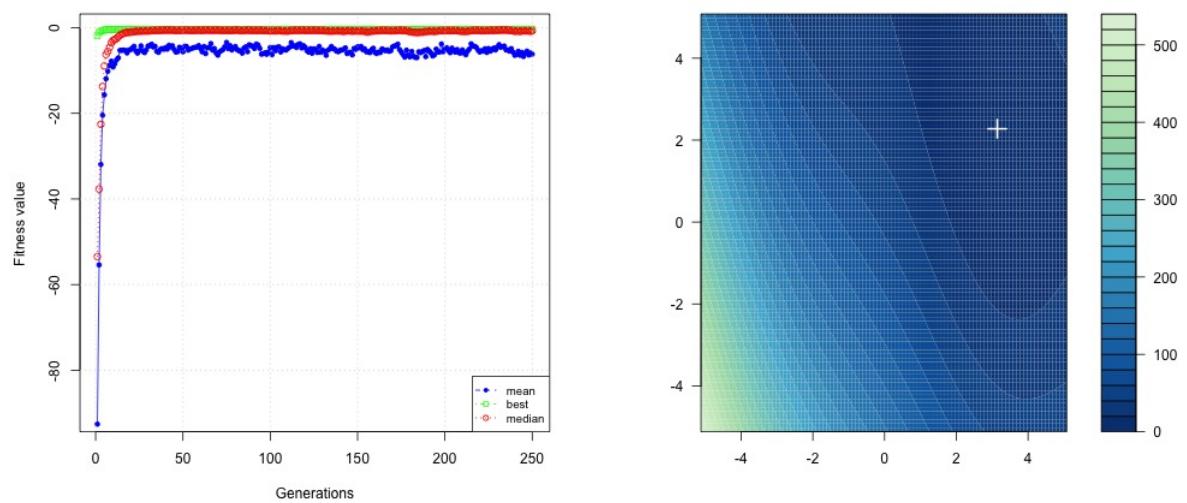
Rysunek 89: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e0.05



Rysunek 90: Test optymalizacji GA Branin p50 i150 c0.8 m0.1 e0.05



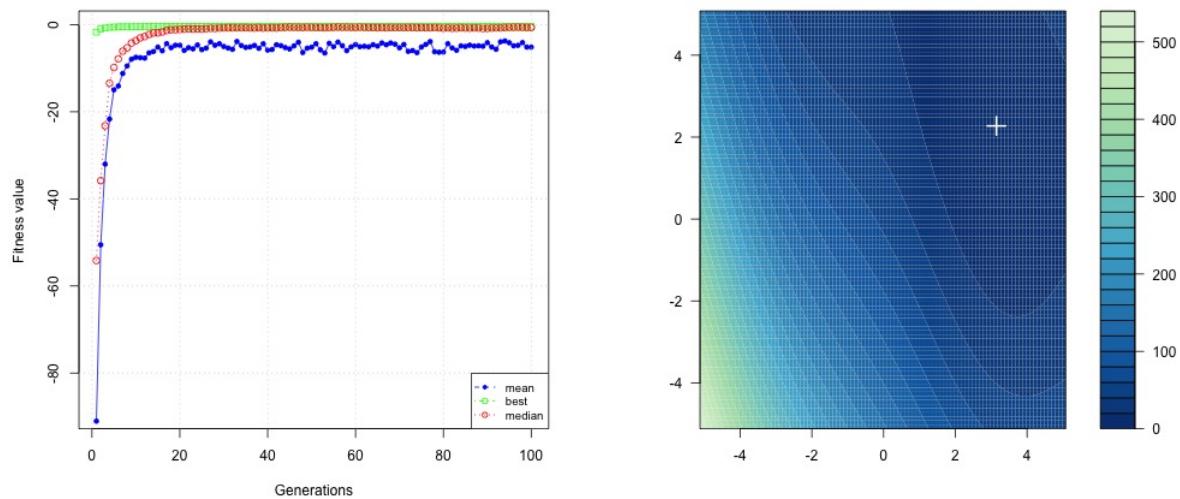
Rysunek 91: Test optymalizacji GA Branin p50 i200 c0.8 m0.1 e0.05



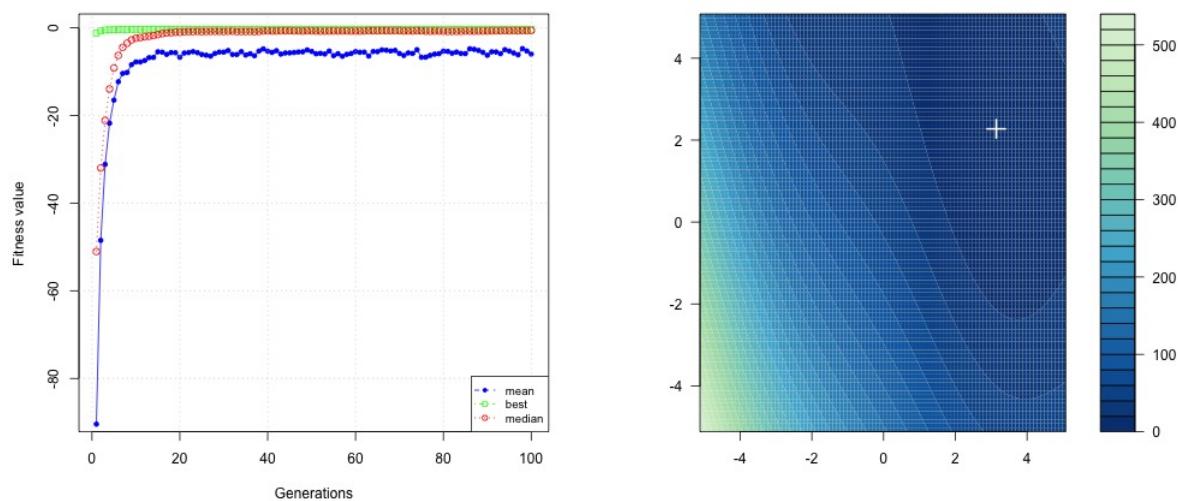
Rysunek 92: Test optymalizacji GA Branin p50 i250 c0.8 m0.1 e0.05

5.4.5 Modyfikacja parametru rozmiaru populacji

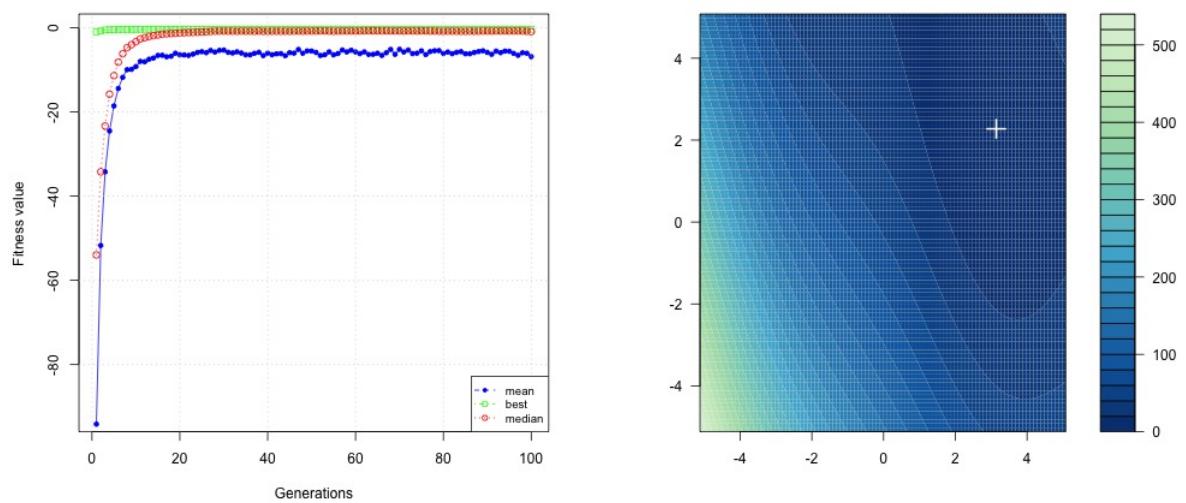
Funkcja Branin nie posiada wielu lokalnych minimów tak jak w jest to w przypadku funkcji Schubert. Dlatego algorytm znajduje optymalne rozwiązanie zawsze i wskazania mediany oraz średniej nie wykazują aby algorytm próbował znaleźć globalnego minimum w miejscu gdzie występuje lokalne minimum.



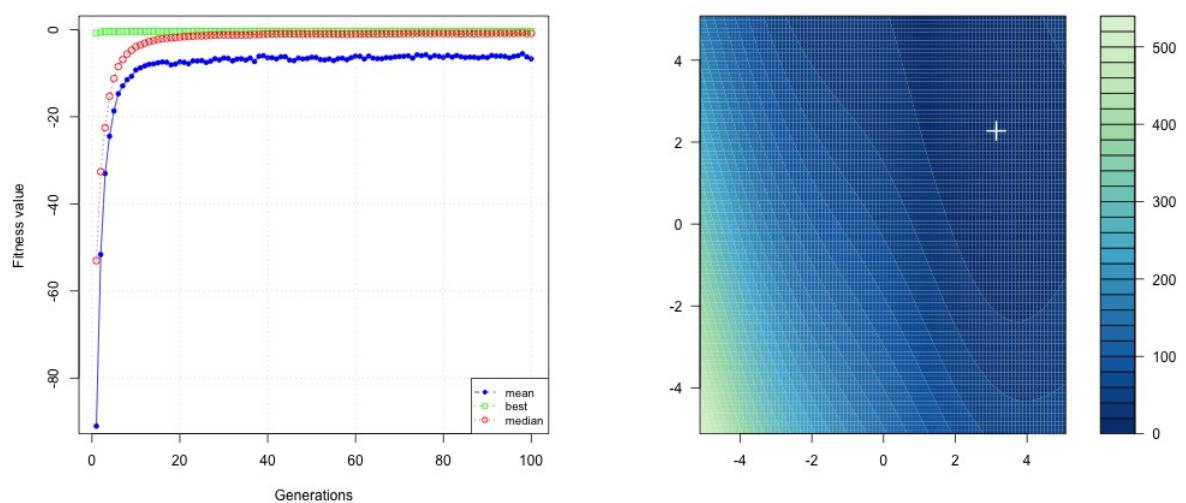
Rysunek 93: Test optymalizacji GA Branin p50 i100 c0.8 m0.1 e0.05



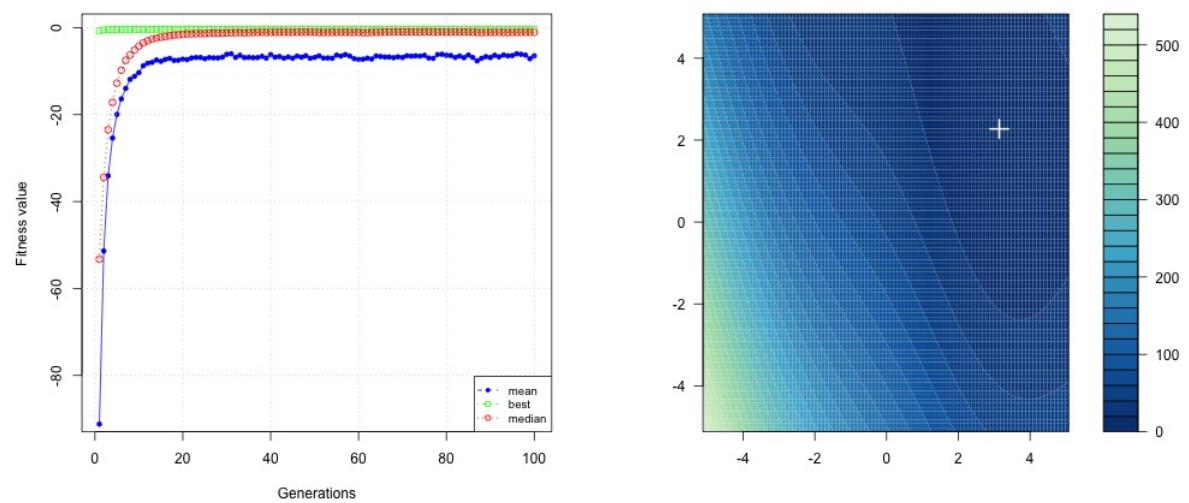
Rysunek 94: Test optymalizacji GA Branin p100 i100 c0.8 m0.1 e0.05



Rysunek 95: Test optymalizacji GA Branin p150 i100 c0.8 m0.1 e0.05



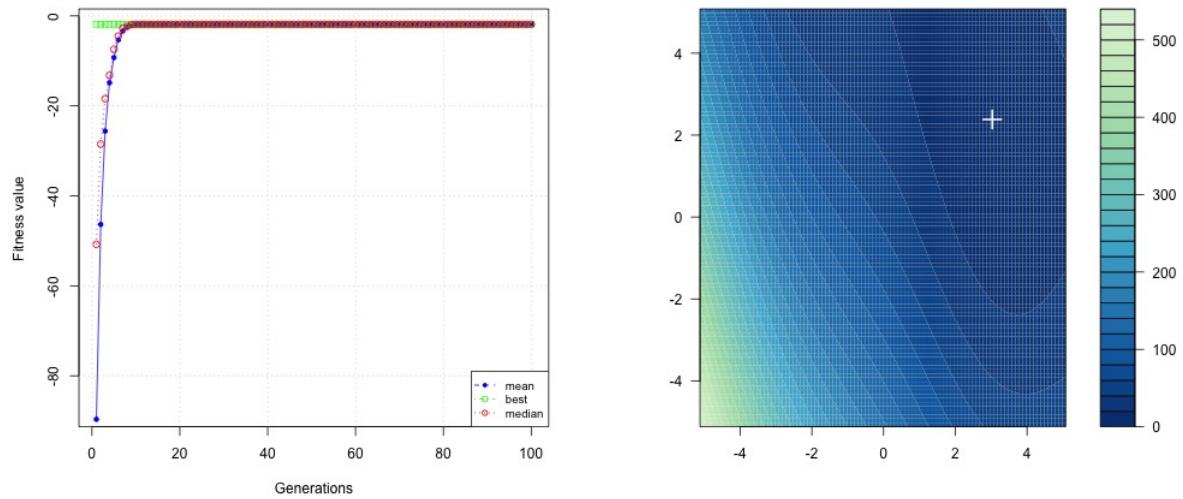
Rysunek 96: Test optymalizacji GA Branin p200 i100 c0.8 m0.1 e0.05



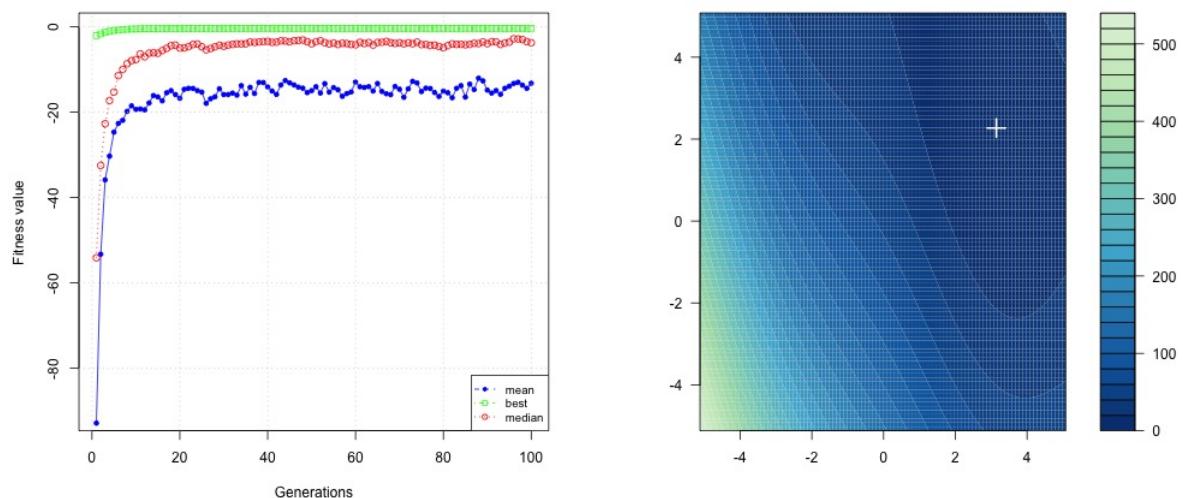
Rysunek 97: Test optymalizacji GA Branin p250 i100 c0.8 m0.1 e0.05

5.5 Jednoczesna modyfikacja parametru krzyżowania i mutacji

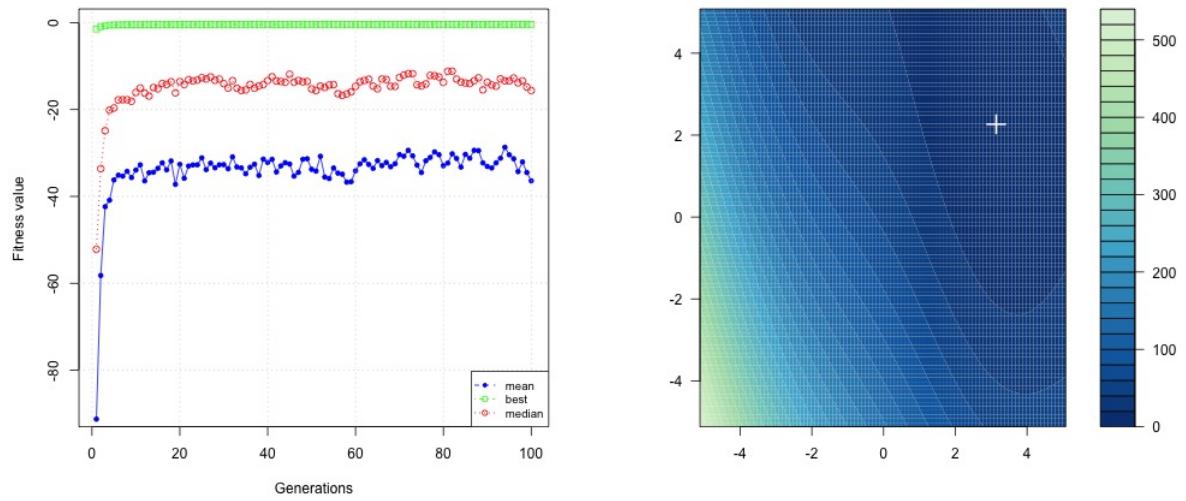
Wartości średniej i mediany są gorsze wraz ze wzrostem obydwu parametrów. W przypadku jednoczesnej modyfikacji parametru krzyżowania i mutacji krzywe rozwiązań mają wartości podobne.



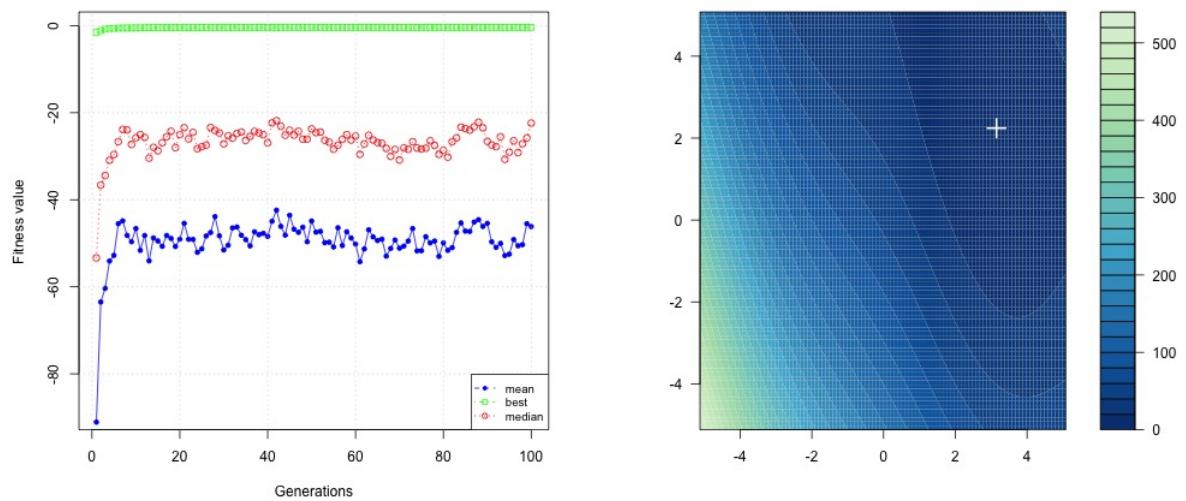
Rysunek 98: Test optymalizacji GA Branin p050 i100 c0.0 m0.0 e0.05



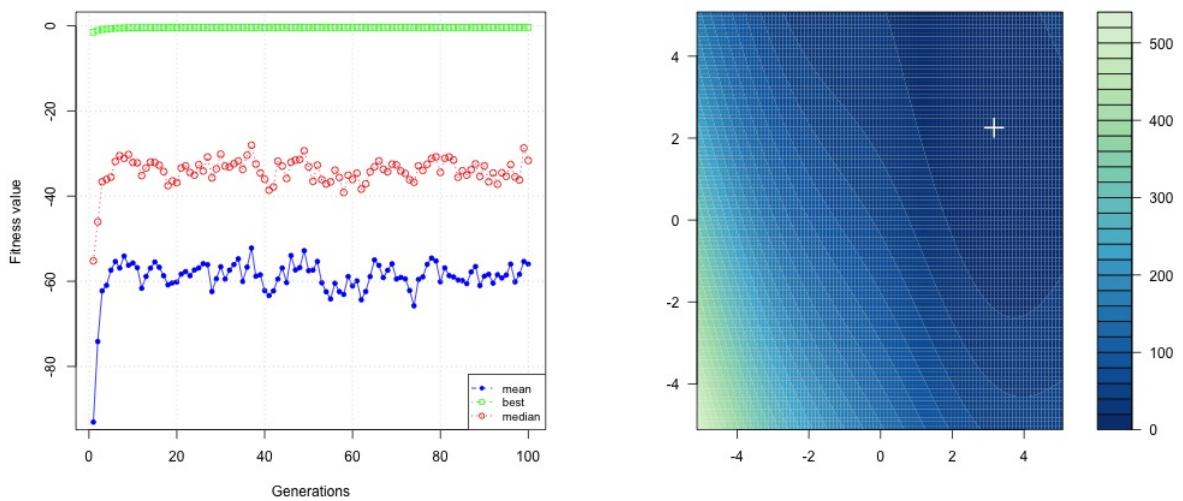
Rysunek 99: Test optymalizacji GA Branin p050 i100 c0.25 m0.25 e0.05



Rysunek 100: Test optymalizacji GA Branin p050 i100 c0.50 m0.50 e0.05



Rysunek 101: Test optymalizacji GA Branin p050 i100 c0.75 m0.75 e0.05



Rysunek 102: Test optymalizacji GA Branin p050 i100 c1.00 m1.00 e0.05

6 Wnioski

Badanie wpływu paramterów na jakość rozwiązań optymalnych poszukiwanych przez algorytm genetyczny jest zadaniem nietrywialnym, ale możliwym do wykonania. Zastosowanie pakietu GA dla języka R pozwoliło skupić się bardziej na testowaniu parametrów, a nie na implementacji algorytmu.

Warto zauważyc, że na jakość rozwiązań ma wpływ nie tylko wartość parametru, ale również funkcja poddawana testom. Dla tych samych zestawów danych wykresy testów różnią się pomiędzy testowanymi funkcjami. W przypadku poszukiwania ekstremum globalnego dla funkcji z wieloma ekstremami lokalnymi i niekorzystnym doborze parametrów algorytm może nie znaleźć poprawnego rozwiązania. Jest to spowodowane stochastycznym doborem populacji początkowej.

7 Literatura

1. Artur Suchwalko, "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

8 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)

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

#uzyte funkcji
fnNames = c("Schubert", "Bohachevsky1", "Branin")

# liczba przebiegow
testInstances = 50

#testowane wartosci parametrow
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)

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

##rysowanie wykresu 3d dla uzytej funkcji
showFunction3dPlot <- function(x1, x2, f) {
  persp3D(x1,
          x2,
          f,
          theta = -50,
          phi = 20,
          color.palette = bl2gr.colors)
}

##rysowanie wykresu temperaturowego dla danej funkcji
showFunctionContour <- function(x1, x2, f) {
  filled.contour(x1, x2, f, color.palette = bl2gr.colors)
}

##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,
          elitismPercentage,
          pcrossover,
          pmutation) {
    #wrapper funkcji
    testFunctionWrapper <- function(x1, x2)
    {
      goTest(par = c(x1, x2), fnName = functionName)
    }

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

    #zapis wykresu 3d dla wybranej funkcji
    jpeg(file = sprintf("%s-3dplot.jpg", path, functionName))
    showFunction3dPlot(x1, x2, f)
    dev.off()
  }
}

```

```

#zapis wykresu temperaturowego dla wybranej funkcji
jpeg(file = sprintf("%s%s-contour.jpg", path, functionName))
showFunctionContour(x1, x2, f)
dev.off()

#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),
    #rozpatrywana przestrzen
    max = c(5.12, 5.12),
    #rozpatrywana przestrzen
    popSize = popSize,
    maxiter = iterationSize,
    elitism = elitsim,
    pcrossover = pcrossover,
    pmutation = pmutation,
    run = iterationSize,
    monitor = FALSE #wylaczenie logowania
  )
  #sumowanie rozwiadan
  tmpGASummary <- GA@summary + tmpGASummary
}
#wyznaczenie sredniej arytmetycznej rozwiadan
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 znalezionej 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 argumentem
invokeTestsWithFunction <- function(functionName) {
  #zmiana wartosci populacji elitarnej
  for (elitsimPercentage in elitePopulationSizes) {
    calculateGA(
      functionName ,
      defaultPopSize ,
      defaultIterationSize ,
      elitsimPercentage ,
      defaultCrossover ,
      defaultMutation
    )
  }

  #zmiana wartosci mutacji
  for (pmutation in mutationSizes) {
    calculateGA(
      functionName ,
      defaultPopSize ,
      defaultIterationSize ,
      defaultElitePopulation ,
      defaultCrossover ,
      pmutation
    )
  }

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

  #jednoczesna zmiana krzyzowania i mutacji
}

```

```

for (pcrossover in crossoverSizes) {
    for (pmutation in mutationSizes) {
        calculateGA(
            functionName ,
            defaultPopSize ,
            defaultIterationSize ,
            defaultElitePopulation ,
            pcrossover ,
            pmutation
        )
    }
}

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

#zmiana liczby populacji
for (popSize in populationSizes) {
    calculateGA(
        functionName ,
        popSize ,
        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))
}

```