

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka

SPECJALNOŚĆ: Inżynieria Systemów Informatycznych (INS)

Inteligencja Obliczeniowa i jej Zastosowania

Sprawozdanie z laboratorium 1 i 2

AUTORZY:
Piotr Chorościn
Dawid Mikowski

PROWADZĄCY:
Dr hab. inż. Olgierd Unold,
prof. uczelni

OCENA PRACY:

SPIS TREŚCI

Spis ilustracji	3
Spis tabel	4
Spis listingów	4
1 Cel ćwiczenia	5
2 Wzory optymalizowanych funkcji z wykresami	6
2.1 Funkcja Schuberta	6
2.2 Funkcja Hosaki	8
3 Zastosowany algorytm optymalizacji	10
3.1 Przegląd zupełny z dokładnością 0,1	10
3.2 Algorytm Genetyczny	10
4 Zastosowane narzędzia implementacji, biblioteki	11
5 Tabele i wykresy z doświadczeń	11
5.1 Badanie wpływu rozmiaru populacji	13
5.1.1 Badanie wpływu rozmiaru populacji na przystosowanie dla funkcji Schuberta	13
5.1.2 Badanie wpływu rozmiaru populacji na przystosowanie dla funkcji Hosaki	13
5.1.3 Obserwacje nt. wpływu liczności populacji na jej średnie przystosowanie oraz przystosowanie najlepszego osobnika	14
5.2 Badanie wpływu liczby pokoleń	15
5.2.1 Badanie wpływu liczby pokoleń na przystosowanie dla funkcji Schuberta	15
5.2.2 Badanie wpływu liczby pokoleń na przystosowanie dla funkcji Hosaki	16
5.2.3 Obserwacje nt. wpływu liczby pokoleń na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika	17
5.3 Badanie wpływu prawdopodobieństwa mutacji	18
5.3.1 Badanie wpływu prawdopodobieństwa mutacji na przystosowanie dla funkcji Schuberta	18
5.3.2 Badanie wpływu prawdopodobieństwa mutacji na przystosowanie dla funkcji Hosaki	19
5.3.3 Obserwacje nt. wpływu prawdopodobieństwa mutacji na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika	20
5.4 Badanie wpływu prawdopodobieństwa krzyżowania	21
5.4.1 Badanie wpływu prawdopodobieństwa krzyżowania na wartość funkcji Schuberta	21
5.4.2 Badanie wpływu prawdopodobieństwa mutacji na przystosowanie dla funkcji Hosaki	22
5.4.3 Obserwacje nt. wpływu prawdopodobieństwa krzyżowania na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika	23
5.5 Badanie wpływu prawdopodobieństwa krzyżowania i mutacji	24
5.6 Badanie wpływu poziomu elitaryzmu	26
5.6.1 Badanie wpływu elitaryzmu na przystosowanie do funkcji Schuberta	26
5.6.2 Badanie wpływu elitaryzmu na przystosowanie do funkcji Hosaki	26

5.6.3	Obserwacje nt. wpływu elitaryzmu na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika.....	27
6	Wnioski	28
6.1	Wnioski nt. algorytmu genetycznego	28
6.2	Wnioski nt. implementacji i języka R	29
7	Kod z komentarzem.....	30
7.1	Przegląd najważniejszych fragmentów kodu	30
7.2	Pełny kod programu	33
8	Literatura	37

SPIS ILUSTRACJI

Rysunek 6.1 Przebieg zmienności wartości funkcji dopasowania najlepszego osobnika i średniej dla całej populacji w zależności od pokolenia dla przykładowego uruchomienia algorytmu genetycznego dla funkcji Hosaki.	12
Rysunek 6.2 Przebieg dla innego uruchomienia (też dla funkcji Hosaki).....	12
Rysunek 6.3 Zależność wartości funkcji celu Schuberta od liczebności populacji.....	13
Rysunek 6.4 Zależność wartości funkcji celu Hosaki od liczebności populacji	13
Rysunek 6.5 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od liczby pokoleń.....	15
Rysunek 6.6 Powiększenie wykresu dla najlepszego osobnika	15
Rysunek 6.7 Powiększenie wykresu dla średniej z całej populacji	15
Rysunek 6.8 Zależność wartości funkcji celu Hosaki dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od liczby pokoleń.....	16
Rysunek 6.9 Powiększenie wykresu dla najlepszego osobnika	16
Rysunek 6.10 Powiększenie wykresu dla średniej z całej populacji	16
Rysunek 6.11 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa mutacji.	18
Rysunek 6.12 Powiększenie wykresu dla najlepszego osobnika	18
Rysunek 6.13 Powiększenie wykresu dla średniej z całej populacji	18
Rysunek 6.14 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa mutacji.	19
Rysunek 6.15 Powiększenie wykresu dla najlepszego osobnika	19
Rysunek 6.16 Powiększenie wykresu dla średniej z całej populacji	19
Rysunek 6.17 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa krzyżowania.....	21
Rysunek 6.18 Powiększenie wykresu dla najlepszego osobnika	21
Rysunek 6.19 Powiększenie wykresu dla średniej z całej populacji	22
Rysunek 6.20 Zależność wartości funkcji celu Hosaki dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa krzyżowania.....	22
Rysunek 6.21 Powiększenie wykresu dla najlepszego osobnika	23
Rysunek 6.22 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od wartości parametru „elitism”	26
Rysunek 6.23 Zależność wartości funkcji celu Hosaki dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od wartości parametru „elitism”	26
Rysunek 6.24 Powiększenie wykresu dla najlepszego osobnika.	27

SPIS TABEL

Tabela 6.1 Wartości parametrów i zmiennych przyjęte w badaniach	11
Tabela 6.2 Poszczególne osobniki populacji w 1. i 60. pokoleniu w zależności od wartości prawdopodobieństw: mutacji i krzyżowania	24

SPIS LISTINGÓW

Listing 8.1 Kod funkcji objective.fun.of	30
Listing 8.2 Kod funkcji objective.fun.get.....	30
Listing 8.3 Kod funkcji objective.fun.plot	31
Listing 8.4 Kod funkcji GA.run.iterations.....	31
Listing 8.5 Kod funkcji GA.run.experiment	31
Listing 8.6 Kod funkcji GA.run.experiment.list.....	32
Listing 8.7 Wywołanie funkcji GA.run.experiment.....	32
Listing 8.8 Funkcja GA.run.once	33
Listing 8.9 Skrypt global_opt.R	33
Listing 8.10 Skrypt global_opt.R	35
Listing 8.11 Skrypt main.R	36

1 CEL ĆWICZENIA

Celem ćwiczenia jest:

- zapoznanie się z metaheurystyką algorytmu genetycznego,
- doświadczalne zbadanie wpływu parametrów oraz heurystyk stosowanych w algorytmie genetycznym na zbieżność funkcji celu,
- zastosowanie algorytmu genetycznego do optymalizacji wybranych funkcji celu.

2 WZORY OPTYMALIZOWANYCH FUNKCJI Z WYKRESAMI

2.1 FUNKCJA SCHUBERTA

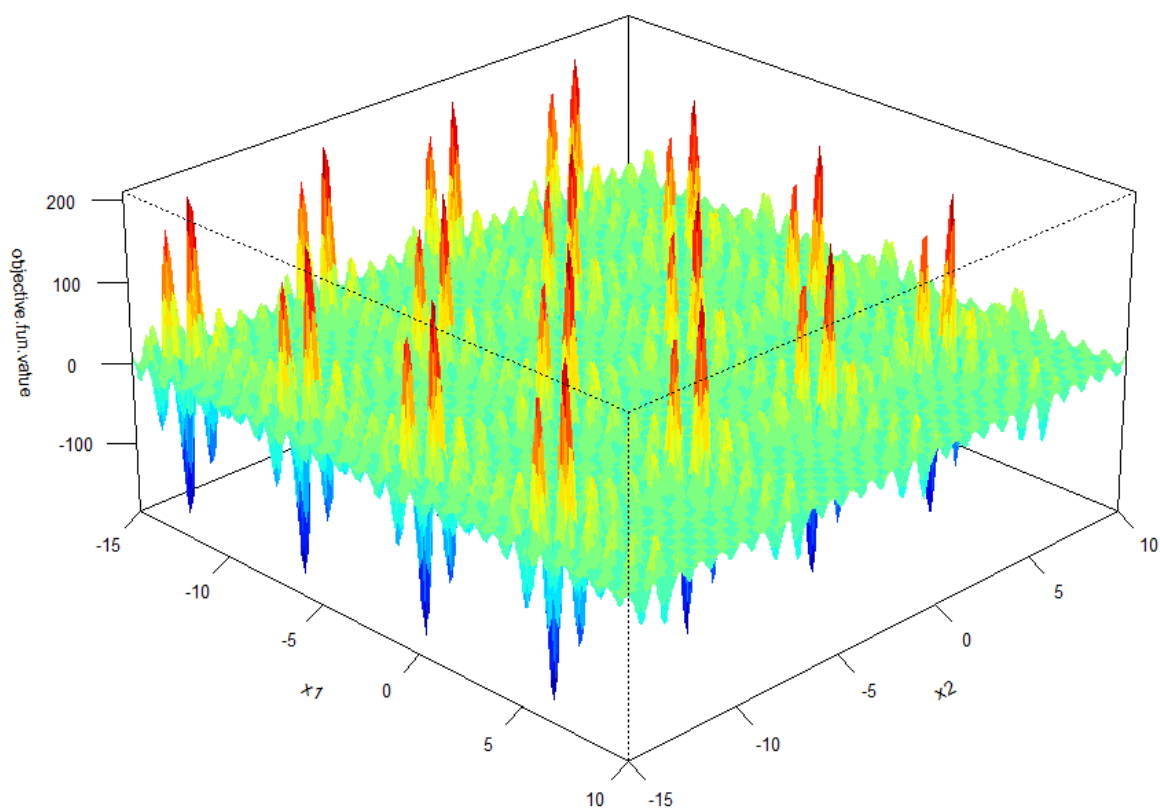
Funkcja Schuberta określona jest poniższym wzorem [1]:

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

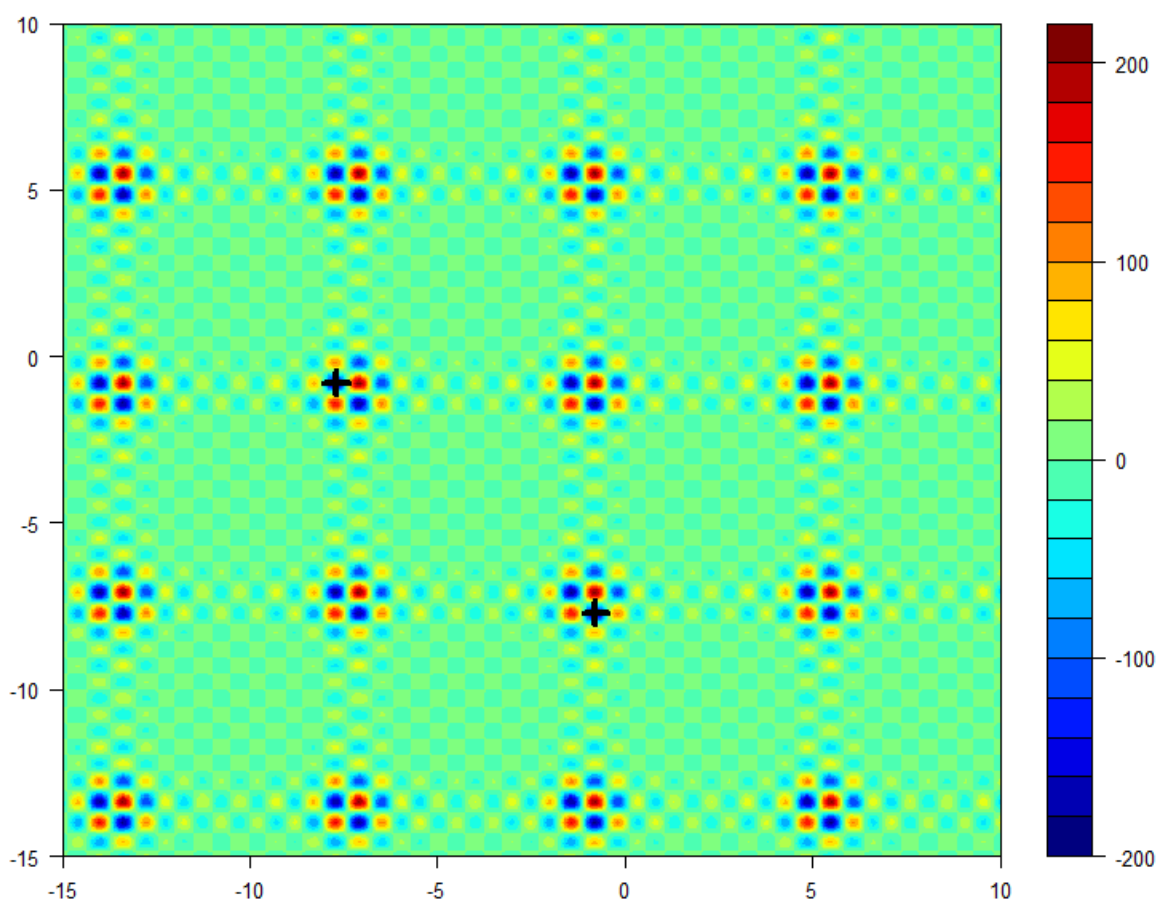
Wykres funkcji Schuberta, wygenerowany dla następujących wartości zmiennych:

- $x_1 \in [-15, 10]$ z krokiem próbkowania 0,1,
- $x_2 \in [-15, 10]$ z krokiem próbkowania 0,1,

Są to domyślne zakresy zmiennych z pakietu *globalOptTest* dla funkcji Schuberta.



Powyższy trójwymiarowy wykres przedstawiony na płaszczyźnie poprzez rzut z góry przedstawiono na poniższym rysunku.



Zgodnie z informacjami z pakietu *GlobalOptTest*, funkcja Schuberta w minimum globalnym przyjmuje wartość **-186,7309**.

$$f(\mathbf{x}^*) = -186,7309$$

W rozpatrywanym zakresie znaleziono 2 minima globalne, w których funkcja celu przyjmuje tę wartość. Są to (z dokładnością do 0,1):

$$\mathbf{x}_1^* = (-0,8; -7,7)$$

$$\mathbf{x}_2^* = (-7,7; -0,8)$$

Ekstrema globalne oznaczono na rysunku czarnymi krzyżykami.

Ekstrema globalne uzyskano w tym przypadku za pomocą pakietu *GlobalOptTest*. Podejście takie odpowiada przeglądowi zupełnemu, w którym dla każdej wartości zmiennych obliczana jest wartość funkcji. Dlatego jednym z celów ćwiczenia jest zastosowanie (w późniejszym etapie) algorytmu genetycznego, który pozwoli operować jedynie na populacji o ograniczonym rozmiarze.

2.2 FUNKCJA HOSAKI

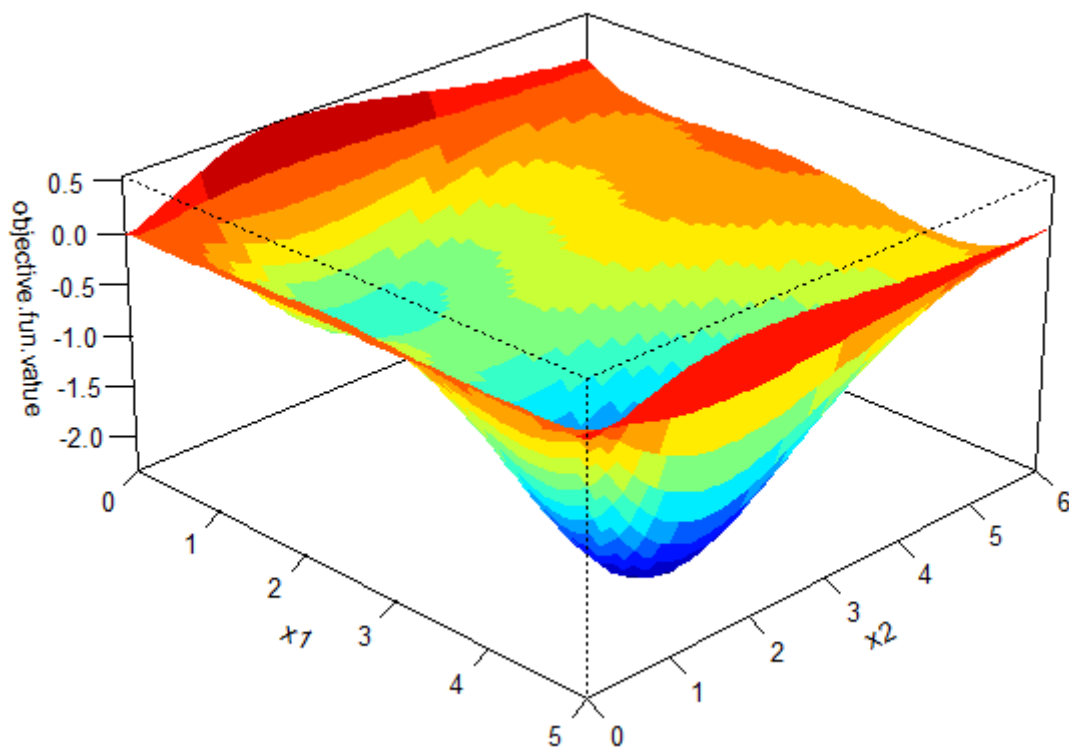
Funkcja Hosaki określona jest poniższym wzorem [2]:

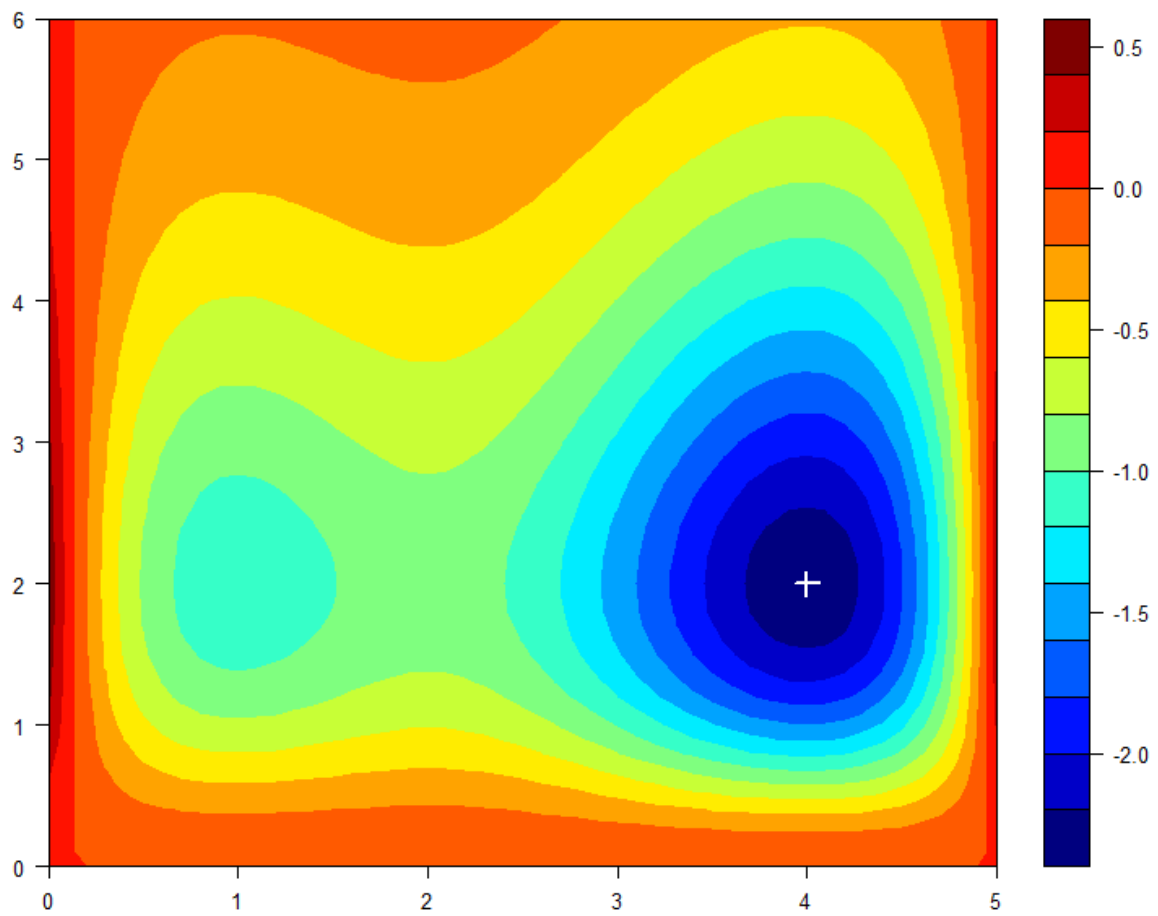
$$f(x_1, x_2) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right)x_2^2e^{-x_1}$$

Wykres funkcji Schuberta, wygenerowany dla następujących wartości zmiennych:

- $x_1 \in [0, 5]$ z krokiem próbkowania 0,1,
- $x_2 \in [0, 6]$ z krokiem próbkowania 0,1,

Są to domyślne zakresy zmiennych z pakietu *globalOptTests* dla funkcji Hosaki.





Zgodnie z informacjami z pakietu *GlobalOptTest*, funkcja Hosaki w minimum globalnym przyjmuje wartość **-2,3458**.

$$f(\mathbf{x}^*) = -2,3458$$

dla $\mathbf{x}^* = (4; 2)$ (z dokładnością do 0,1).

Ekstremum globalne oznaczono na rysunku białym krzyżykiem.

3 ZASTOSOWANY ALGORYTM OPTYMALIZACJI

Wartości funkcji w ekstremum globalnym przechowywane są w pakiecie globalOptTest. Jednak żeby dotrzeć do tego ekstremum (wartości zmiennych odpowiadających wartości minimalnej/maksymalnej funkcji celu) należałoby wykonać przegląd zupełny. Jest bardzo kosztowne obliczeniowo, a ponadto nie zawsze możliwe – np. w przypadku wartości ciągłych zmiennych. Stąd do wyznaczenia ekstremów globalnych w rozdziale 2 zastosowano przegląd, zakładając próbkowanie wartości zmiennych z krokiem 0,1.

W tym rozdziale natomiast zastosowano algorytm genetyczny.

3.1 PRZEGLĄD Z DOKŁADNOŚCIĄ 0,1

Ponieważ wymagane było wygenerowanie wykresu funkcji, konieczne było wyznaczenie wystarczającej liczby punktów, pozwalającej utworzyć taki wykres. Punkty te wygenerowano w ramach domyślnych zakresów wartości zmiennych z krokiem 0,1 pomiędzy kolejnymi wartościami. Dla tak wyznaczonych punktów obliczono wartość funkcji celu, co pozwoliło na narysowanie funkcji. Znalezienie współrzędnych ekstremum globalnego w takim przypadku sprowadza się do odnalezienia takiego punktu, gdzie wartość funkcji celu będzie optymalna (minimalna / maksymalna). Jednak ponieważ przegląd zupełny zazwyczaj jest czasochłonny lub wręcz nie możliwy – dlatego do optymalizacji złożonych funkcji celu, rozpatrywanych w tym zadaniu zastosowany został algorytm genetyczny.

3.2 ALGORYTM GENETYCZNY

W algorytmie genetycznym zamiast rozpatrywać wszystkie możliwe kombinacje wartości atrybutów (tak jak w przypadku przeglądu zupełnego) operuje się na populacji osobników o ograniczonej liczebności [3].

W przypadku badanych funkcji, osobnik kodowany będzie jako wektor współrzędnych punktu – sam osobnik zaś jest punktem w przestrzeni. Przystosowanie osobnika określa wartość funkcji celu w tym punkcie.

Początkowa populacja może być losowa. Począwszy od niej, dla każdego kolejnego pokolenia aplikowane są operatory genetyczne, mające na celu uzyskanie kolejnego pokolenia o lepszym przystosowaniu aniżeli poprzednie. Przystosowanie mierzone jest funkcją przystosowania (ang. *fitness function*). W zadaniach optymalizacji, np. maksymalizacji funkcji jest to po prostu funkcja celu.

Operatory genetyczne aplikowane są w pętli aż do osiągnięcia określonego limitu pokoleń lub też do momentu stabilizacji populacji – jeśli określoną liczbę kolejnych pokoleń nie następuje poprawa osobników. Operatory te to:

- Operator selekcji,
- Operator krzyżowania,
- Operator mutacji.

Każdy z tych operatorów może być zdefiniowany z użyciem różnych heurystyk. W niniejszym ćwiczeniu ograniczono się do zbadania wpływu parametrów – a nie doboru samych heurystyk.

4 ZASTOSOWANE NARZĘDZIA IMPLEMENTACJI, BIBLIOTEKI

Wykorzystano następujące paczki języka R:

- *GA* – zawierająca implementację algorytmu genetycznego,
- *GlobalOptTest* – zawierająca funkcje celu wraz z wartościami ich ekstremów.

Środowisko implementacyjne:

- Interpreter języka R w wersji 3.6.1 z pakietu *Anaconda*,
- Środowisko IDE *IntelliJ* z wtyczką do języka R.

5 TABELE I WYKRESY Z DOŚWIADCZEŃ

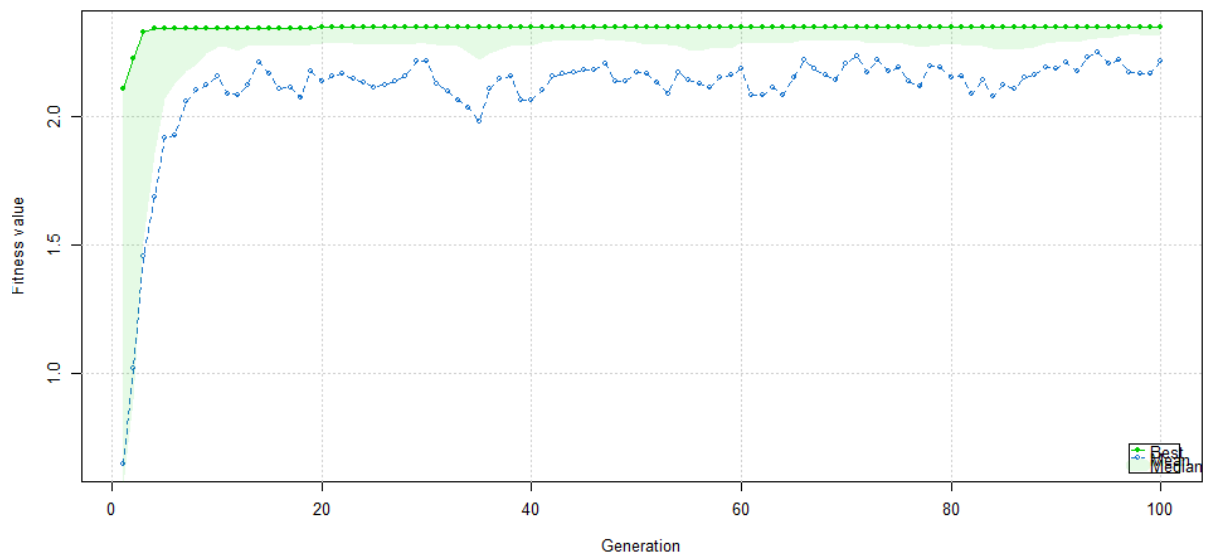
Badania przeprowadzono dla algorytmu genetycznego przy różnych wartościach parametrów. Domyślnie parametry badano niezależnie, tzn. w każdym badaniu zmianom podlegała wartość tylko jednego parametru, inne zaś pozostawały stałe (chyba że w opisie badania zaznaczono inaczej).

Wartości badanych parametrów zamieszczono w tabeli. Kolumna *const* odpowiada wartości parametru w eksperymentach, gdzie był on stały, a *var* w eksperymentach gdzie podlegał zmianom (był zmienną).

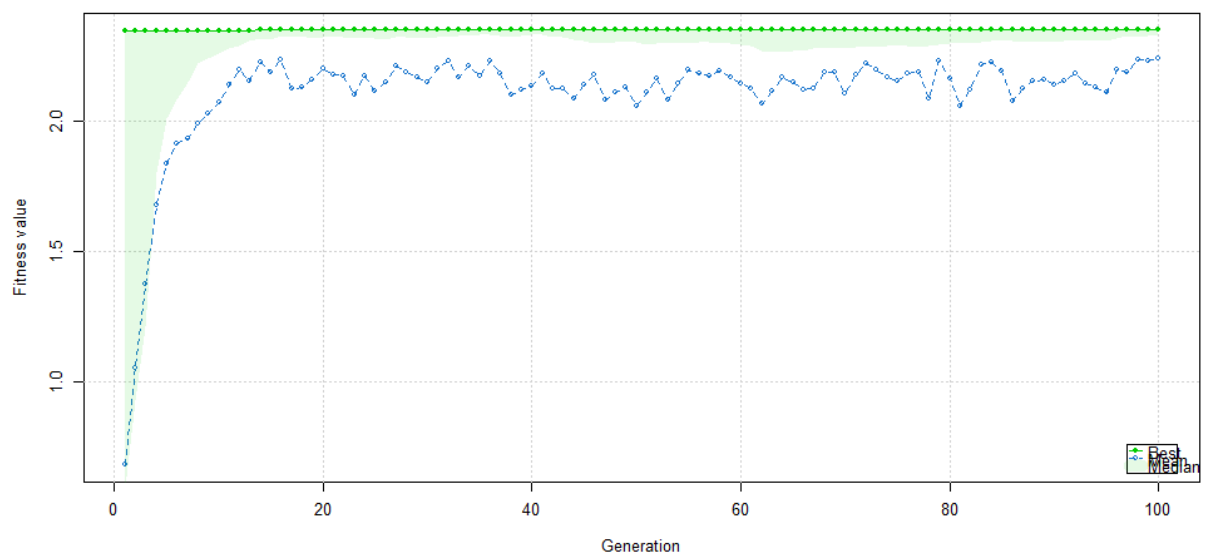
Tabela 5.1 Wartości parametrów i zmiennych przyjęte w badaniach

Nazwa parametru	Const	Var
Liczebność populacji	100	1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 60, 80, 100, 140, 160
Liczba pokoleń	100	1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 60, 80, 100, 140, 160
Elitaryzm	1	0, 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 60, 80, 100
Prawdopodobieństwo mutowania	0,8	[0,0; 1,0] z krokiem 0,05
Prawdopodobieństwo krzyżowania	0,2	[0,0; 1,0] z krokiem 0,05

Wyniki uśredniano dla 10 powtórzeń przebiegu algorytmu genetycznego dla każdej z badanych wartości parametrów. Działanie to było konieczne ponieważ wyniki mogą się różnić w zależności od uruchomienia algorytmu genetycznego, gdyż występuje w nim nie determinizm – np. przy losowym wyborze populacji początkowej. Dla zilustrowania różnic w 2 przykładowych uruchomieniach algorytmu genetycznego zamieszczono wykresy zmian wartości funkcji celu dla tych samych parametrów dla funkcji „Hosaki”. Różnice widoczne są „gołym okiem”.



Rysunek 5.1 Przebieg zmienności wartości funkcji dopasowania najlepszego osobnika i średniej dla całej populacji w zależności od pokolenia dla przykładowego uruchomienia algorytmu genetycznego dla funkcji Hosaki.



Rysunek 5.2 Przebieg dla innego uruchomienia (też dla funkcji Hosaki).

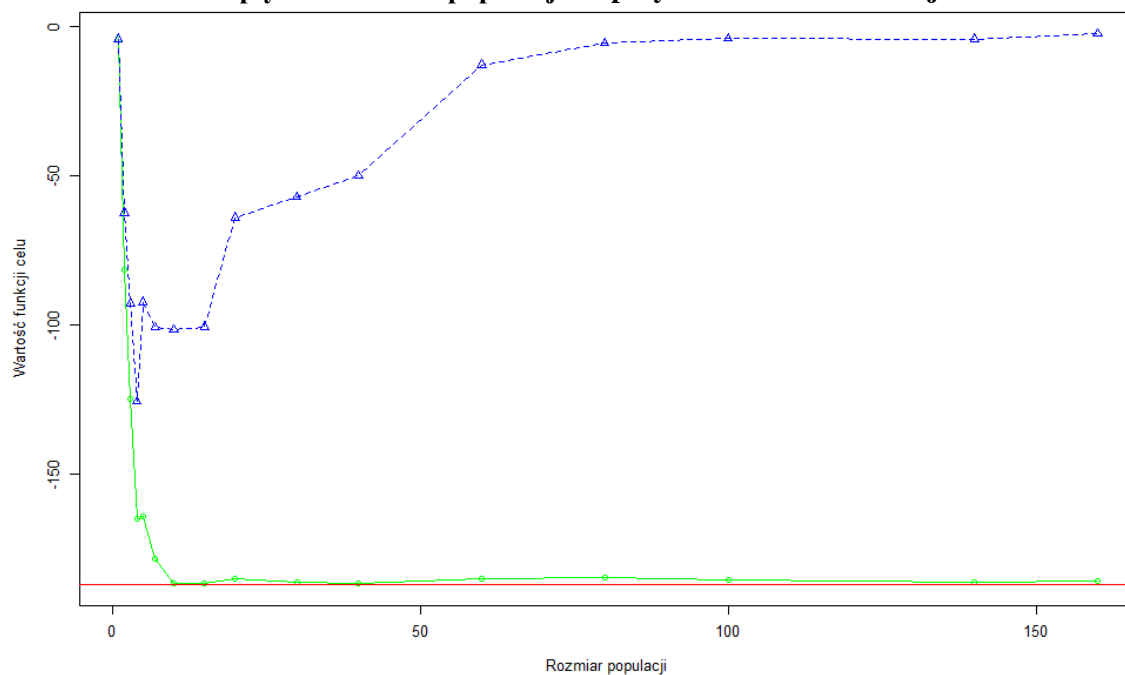
Uwaga: Przyjęto dalej następujące oznaczenia na wykresach (legenda):

- **kolor zielony** – wartość najlepsza w danej populacji,
- **kolor niebieski** – wartość średnia w danej populacji,
- **kolor czerwony** – optimum globalne danej funkcji.

5.1 BADANIE WPŁYWU ROZMIARU POPULACJI

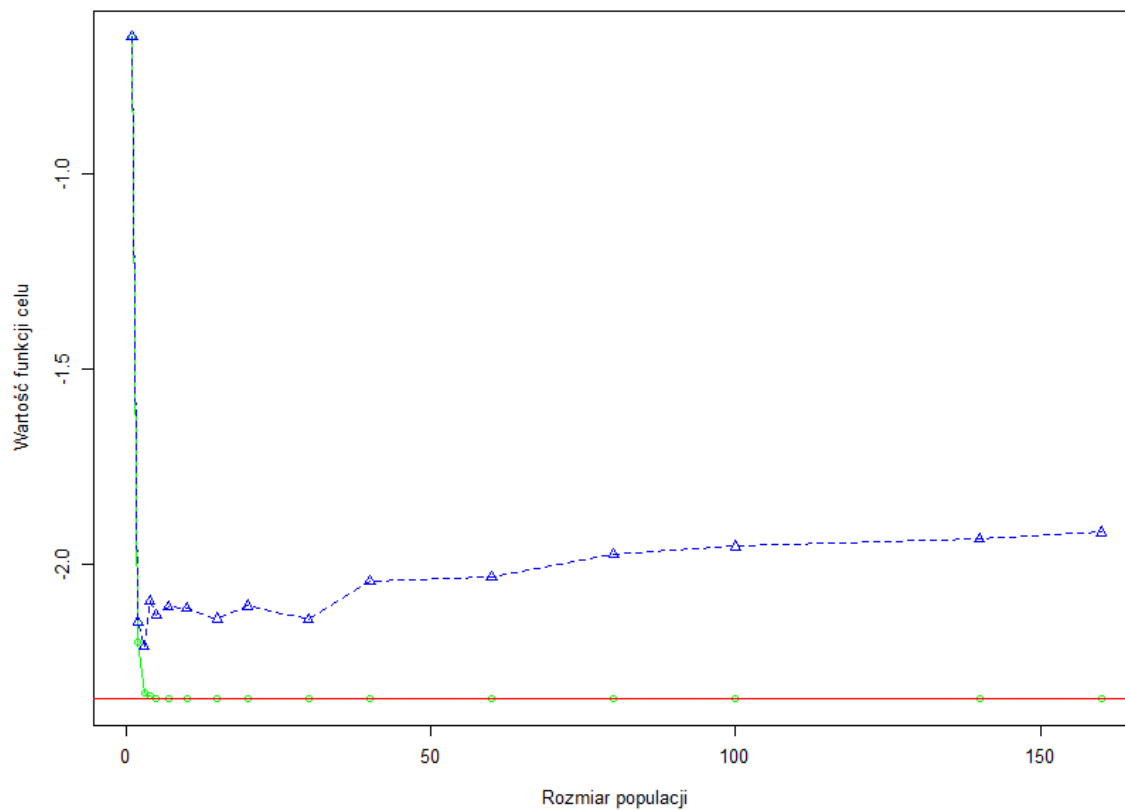
Zmienny rozmiar populacji, pozostałe parametry – stałe. Badania przeprowadzono dla każdej z 2 rozpatrywanych funkcji celu.

5.1.1 Badanie wpływu rozmiaru populacji na przystosowanie dla funkcji Schuberta



Rysunek 5.3 Zależność wartości funkcji celu Schuberta od liczebności populacji

5.1.2 Badanie wpływu rozmiaru populacji na przystosowanie dla funkcji Hosaki



Rysunek 5.4 Zależność wartości funkcji celu Hosaki od liczebności populacji

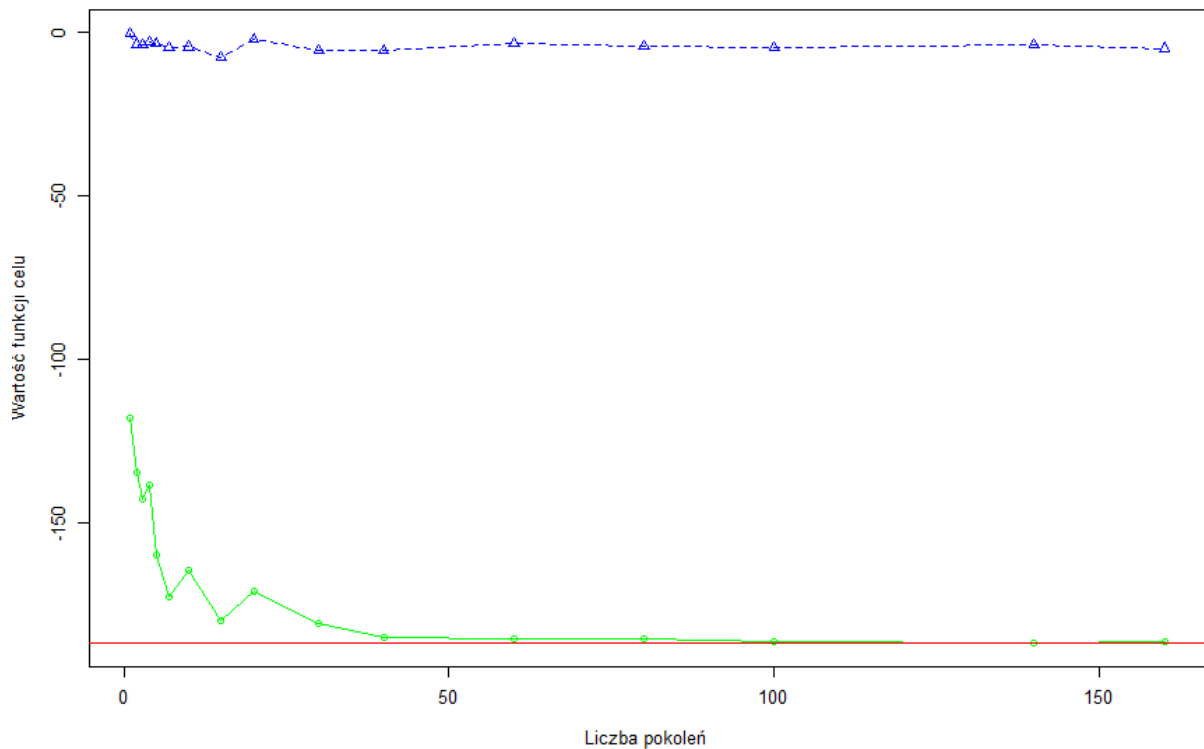
5.1.3 Obserwacje nt. wpływu liczebności populacji na jej średnie przystosowanie oraz przystosowanie najlepszego osobnika

Obserwacje dot. wpływu liczebności populacji na wartość funkcji celu:

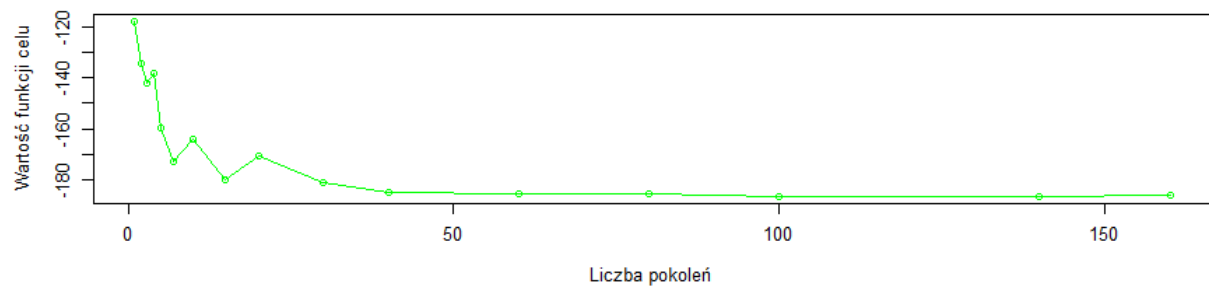
- w przypadku obu funkcji celu dla ustalonych wartości parametru już niewielkie zwiększenie liczebności populacji (do 5 osobników) powoduje odnalezienie ekstremum globalnego,
- średnia jakość osobników w populacji początkowo również bardzo mocno rośnie wraz ze wzrostem liczebności populacji, jednak dla większych liczebności średnia jakość ulega pogorszeniu,
- pogorszenie wartości średniej dla dużych rozmiarów populacji wynika z faktu, że pozostałe parametry, w tym **liczba pokoleń** pozostają stałe. W związku z tym, przy dużej liczbie osobników, większość populacji „nie zdąży” **wyewoluować** przy zbyt niskiej liczbie pokoleń. Natomiast raz znaleziony najlepszy osobnik jest i tak przekazywany do dalszego pokolenia – nie zmienia się praktycznie zatem jakość najlepszego osobnika.

5.2 BADANIE WPŁYWU LICZBY POKOLEŃ

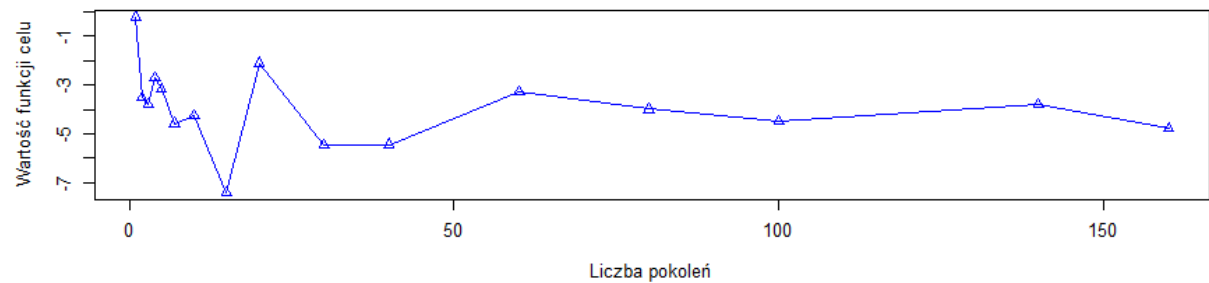
5.2.1 Badanie wpływu liczby pokoleń na przystosowanie dla funkcji Schuberta



Rysunek 5.5 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od liczby pokoleń

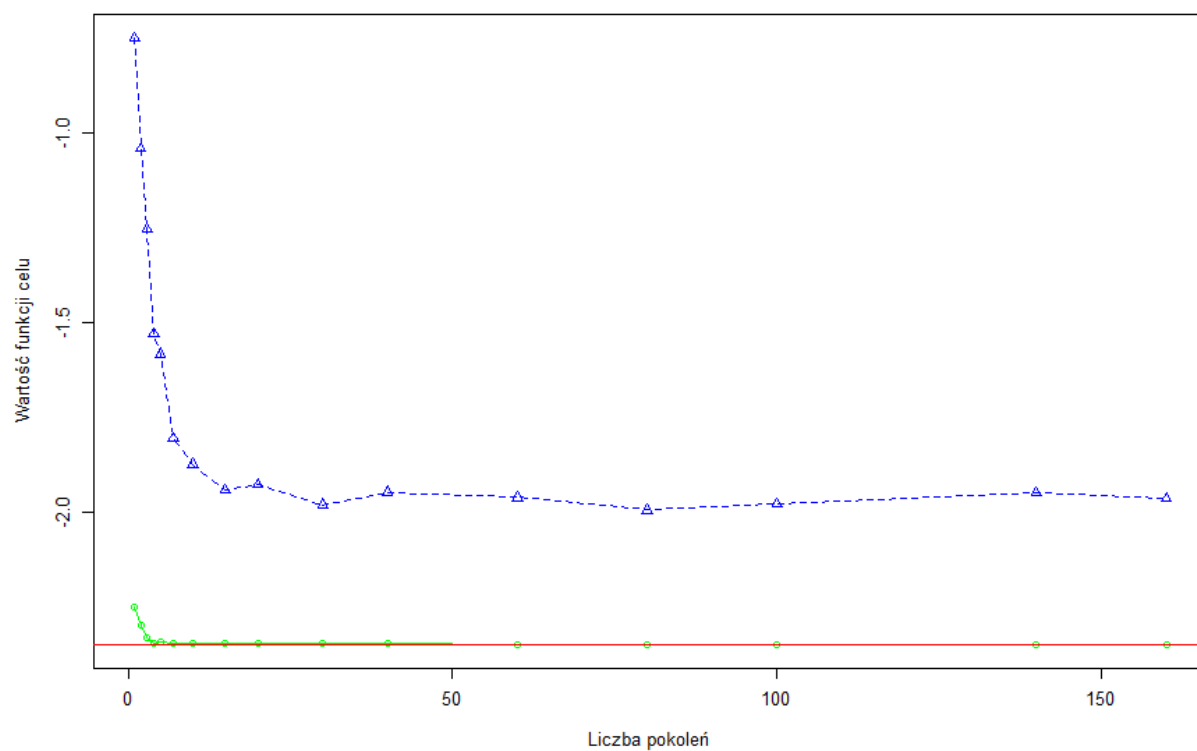


Rysunek 5.6 Powiększenie wykresu dla najlepszego osobnika

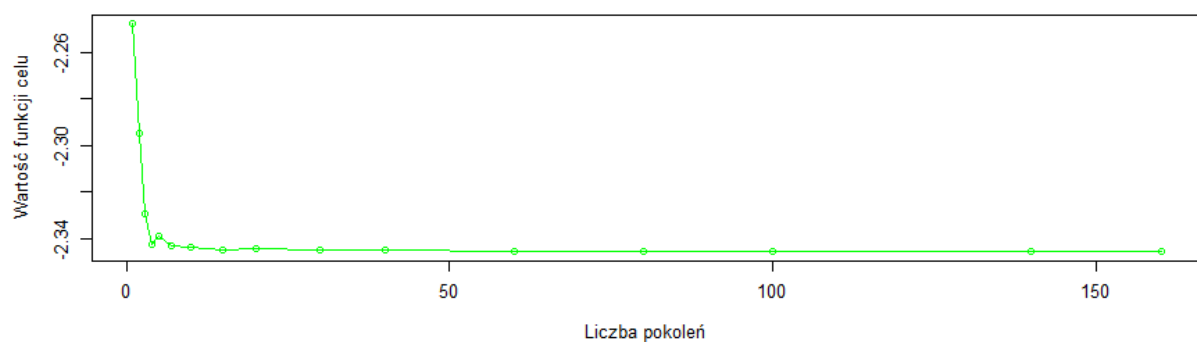


Rysunek 5.7 Powiększenie wykresu dla średniej z całej populacji

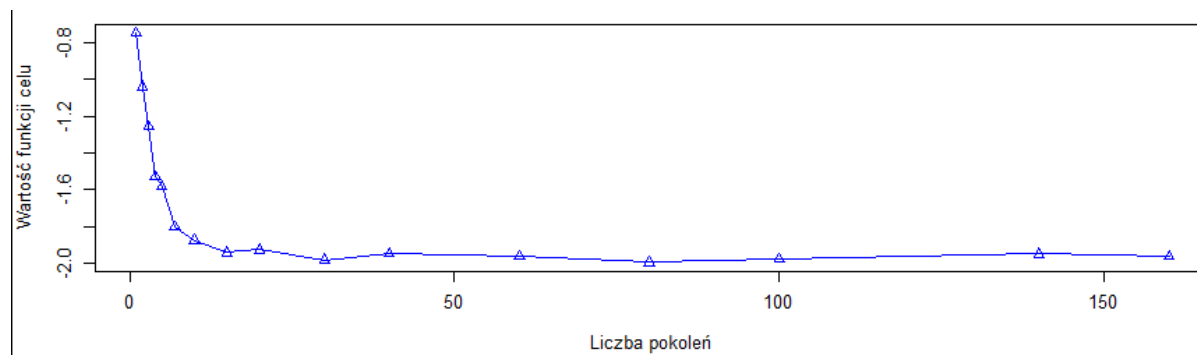
5.2.2 Badanie wpływu liczby pokoleń na przystosowanie dla funkcji Hosaki



Rysunek 5.8 Zależność wartości funkcji celu Hosaki dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od liczby pokoleń



Rysunek 5.9 Powiększenie wykresu dla najlepszego osobnika



Rysunek 5.10 Powiększenie wykresu dla średniej z całej populacji

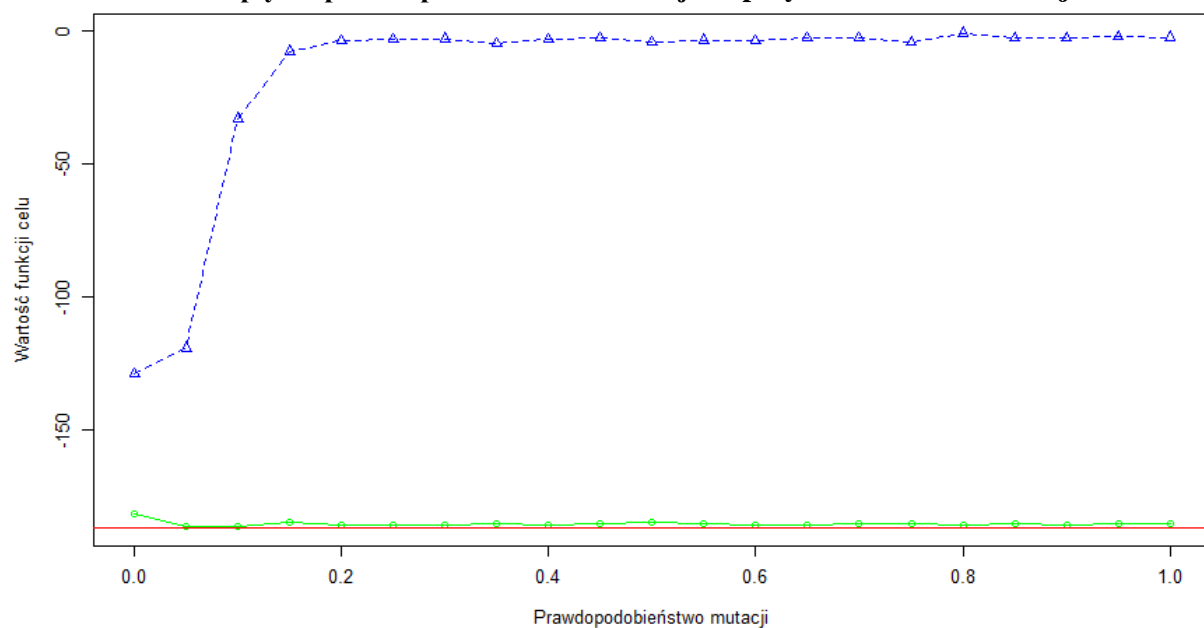
5.2.3 Obserwacje nt. wpływu liczby pokoleń na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika

Na podstawie badań poczyniono następujące obserwacje:

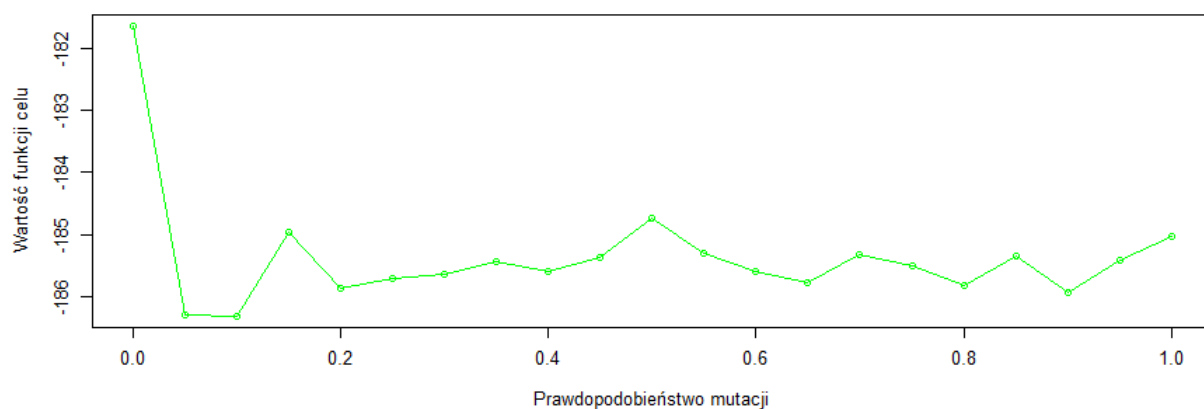
- w przypadku obu funkcji celu dla ustalonych zwiększanie liczby pokoleń powoduje początkowo gwałtowną poprawę średniej jakości populacji jak również jakości najlepszego osobnika. Następnie występuje jednak moment stabilizacji, gdzie w kolejnych pokoleniach nie odnotowuje się już znaczącej poprawy,
- stabilizacja dla najlepszego osobnika może być związana (i w rozpatrywanych przykładach jest) ze znalezieniem ekstremum globalnego funkcji – stąd w kolejnych pokoleniach nie można już poprawić tego wyniku,
- średnia jakość populacji również ulega stabilizacji jednak na niższym (co do wartości bezwzględnej) poziomie aniżeli wartość funkcji przystosowania dla najlepszego osobnika.

5.3 BADANIE WPŁYWU PRAWDOPODOBIENSTWA MUTACJI

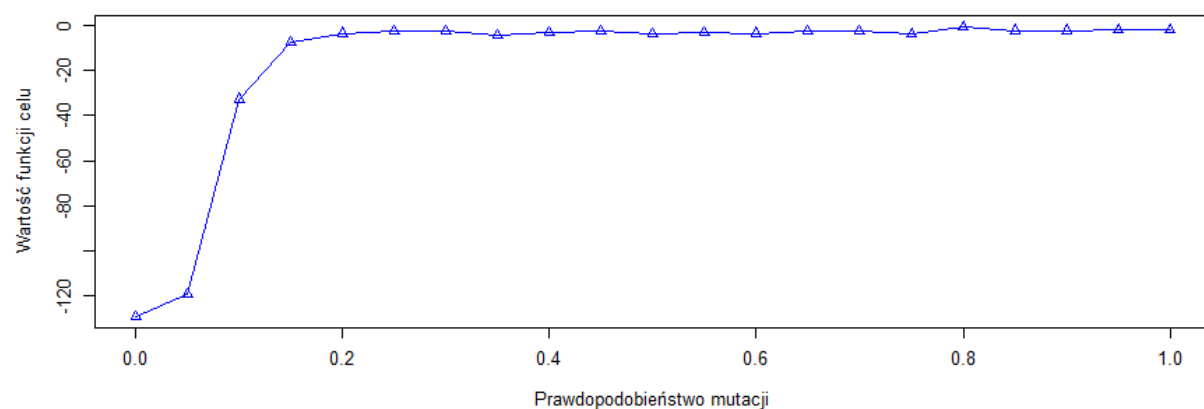
5.3.1 Badanie wpływu prawdopodobieństwa mutacji na przystosowanie dla funkcji Schuberta



Rysunek 5.11 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa mutacji.

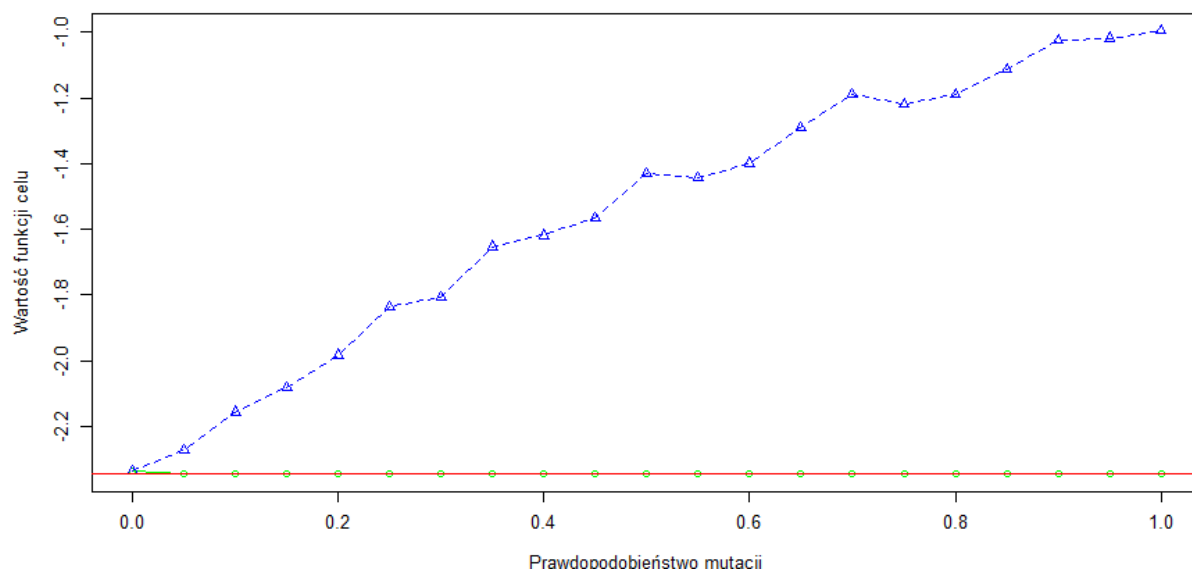


Rysunek 5.12 Powiększenie wykresu dla najlepszego osobnika

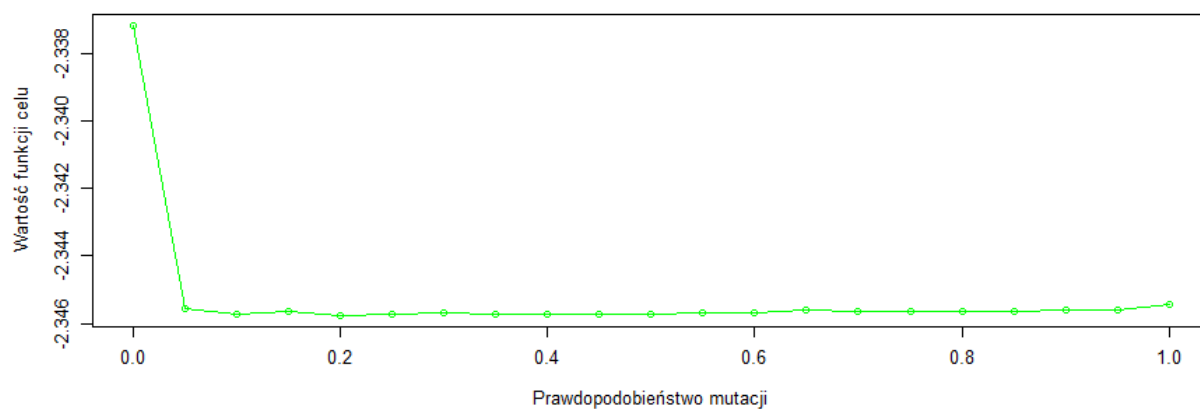


Rysunek 5.13 Powiększenie wykresu dla średniej z całej populacji

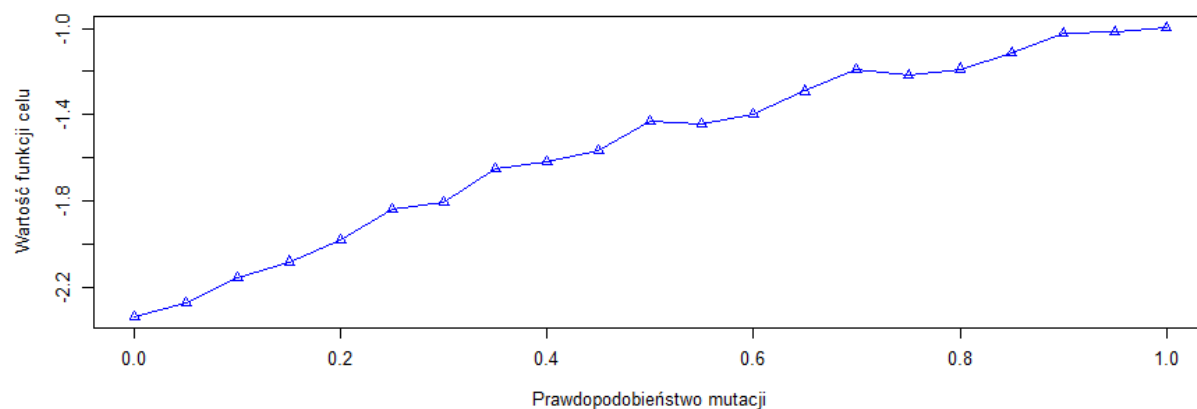
5.3.2 Badanie wpływu prawdopodobieństwa mutacji na przystosowanie dla funkcji Hosaki



Rysunek 5.14 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa mutacji.



Rysunek 5.15 Powiększenie wykresu dla najlepszego osobnika



Rysunek 5.16 Powiększenie wykresu dla średniej z całej populacji

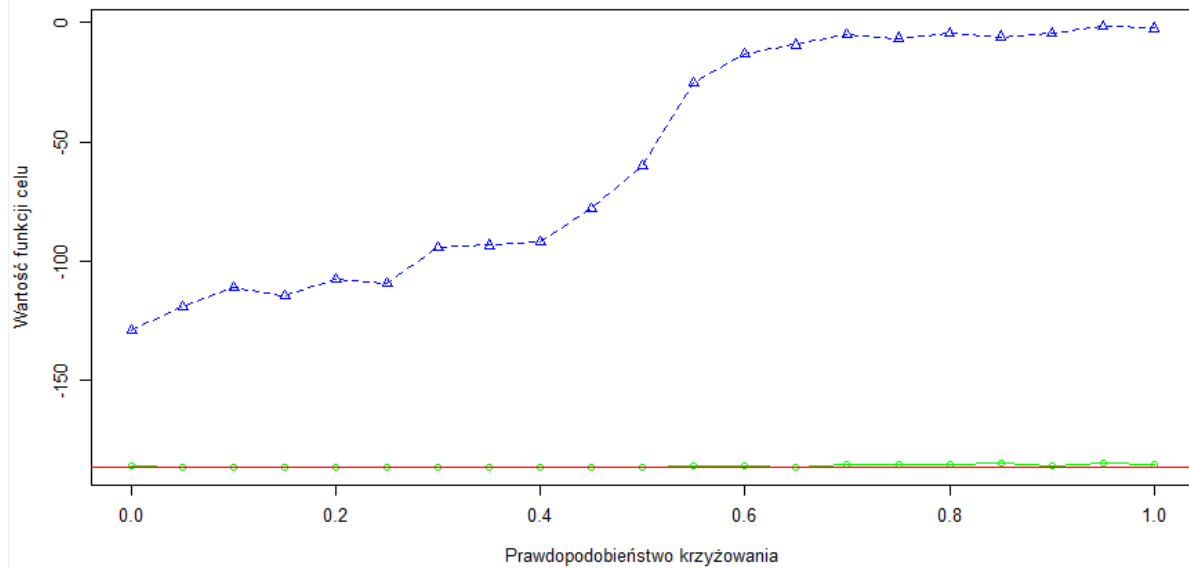
5.3.3 Obserwacje nt. wpływu prawdopodobieństwa mutacji na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika

Na podstawie badań poczyniono następujące obserwacje:

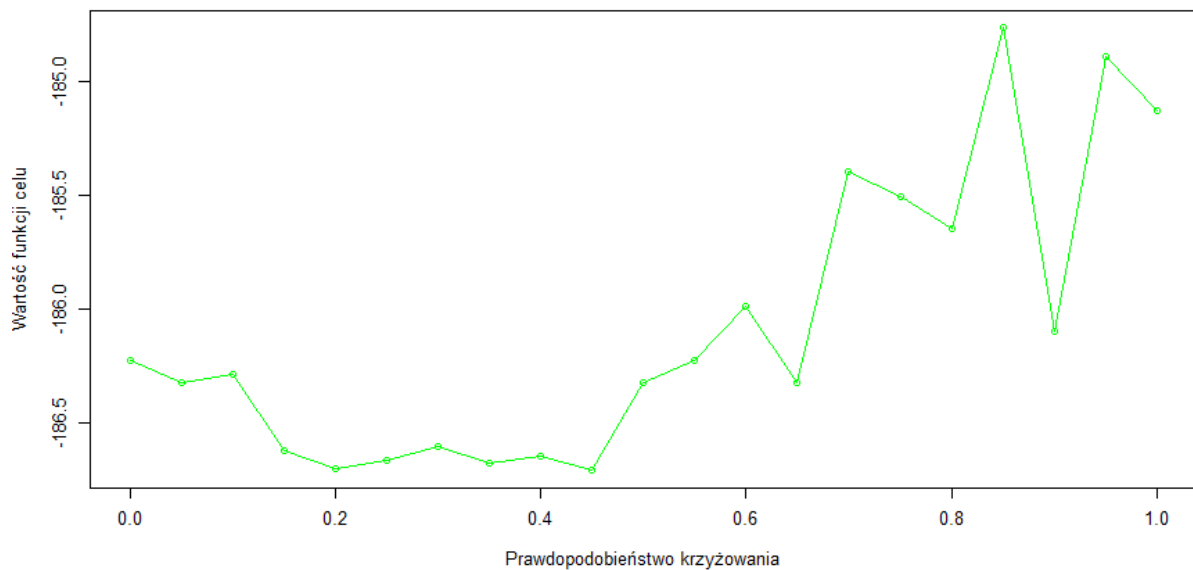
- przystosowanie najlepszego osobnika jest najwyższe w przypadku prawdopodobieństwa mutacji równego ok. 0,1,
- zerowe prawdopodobieństwo mutacji bardzo negatywnie wpływa na przystosowanie najlepszego osobnika – wówczas może on bowiem utknąć w minimum lokalnym – mutacja to znacząca zmiana, która może spowodować „uwolnienie” z minimum lokalnego,
- jednak zwiększanie prawdopodobieństwa mutacji negatywnie wpływa na przystosowanie populacji jako całości (średnie przystosowanie). Pojawiają się bowiem „mutanty” – osobniki po mutacji, których wartość funkcji przystosowania jest znacznie gorsza od pozostałych osobników w populacji,
- rozsądnym wyborem jest zatem prawdopodobieństwo mutacji równe 0,1 – jednocześnie pozwala uniknąć utknięcia w minimum lokalnym, jak również nie psuje zbyt wiele jakości całej populacji.

5.4 BADANIE WPŁYWU PRAWDOPODOBIENSTWA KRZYŻOWANIA

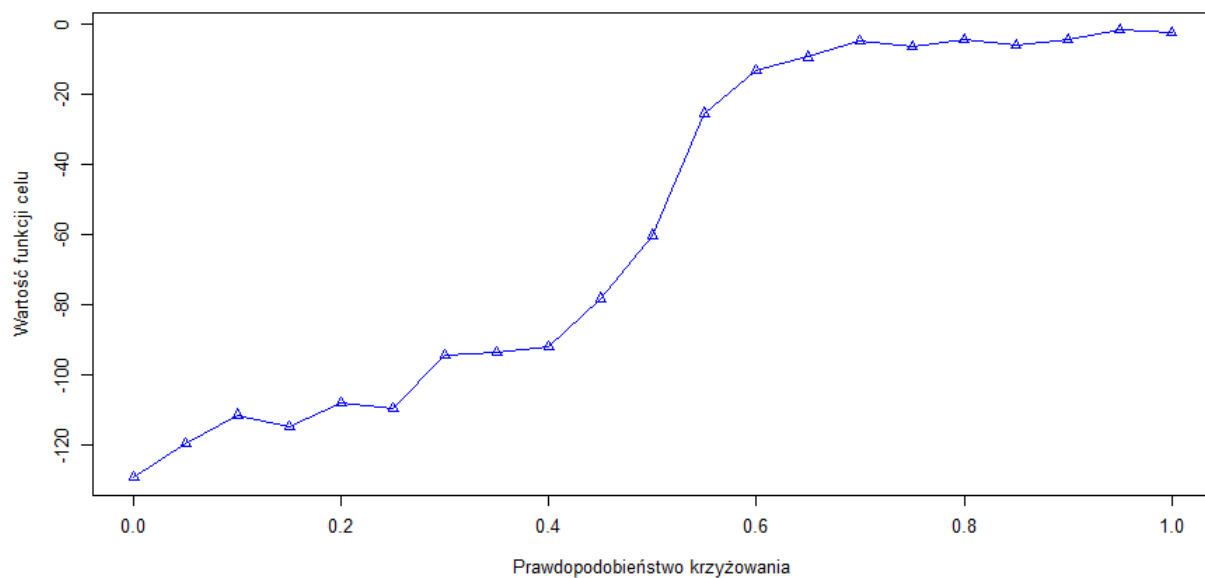
5.4.1 Badanie wpływu prawdopodobieństwa krzyżowania na wartość funkcji Schuberta



Rysunek 5.17 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa krzyżowania

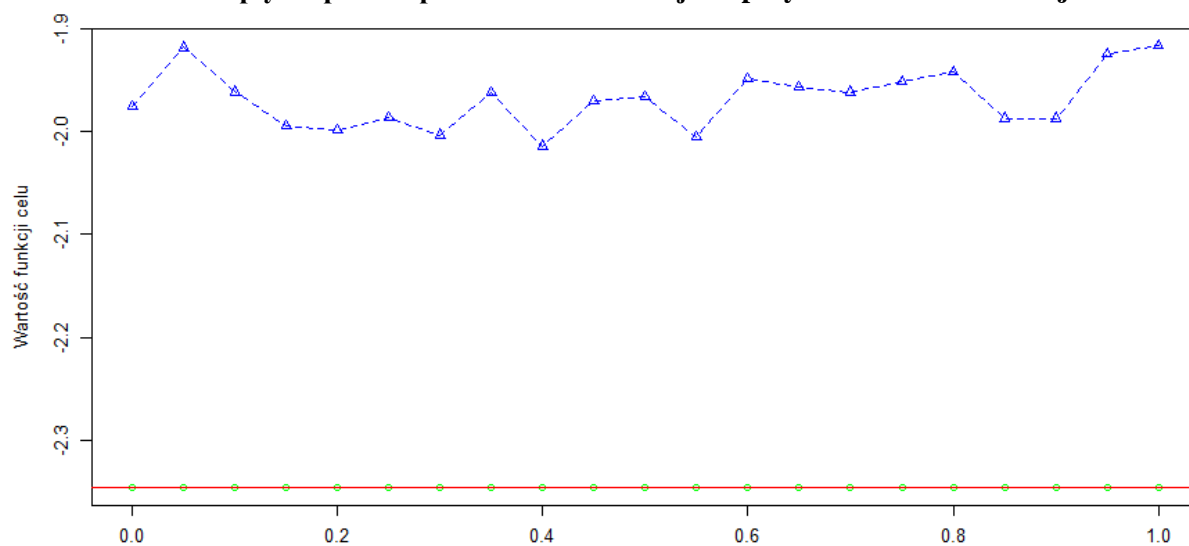


Rysunek 5.18 Powiększenie wykresu dla najlepszego osobnika

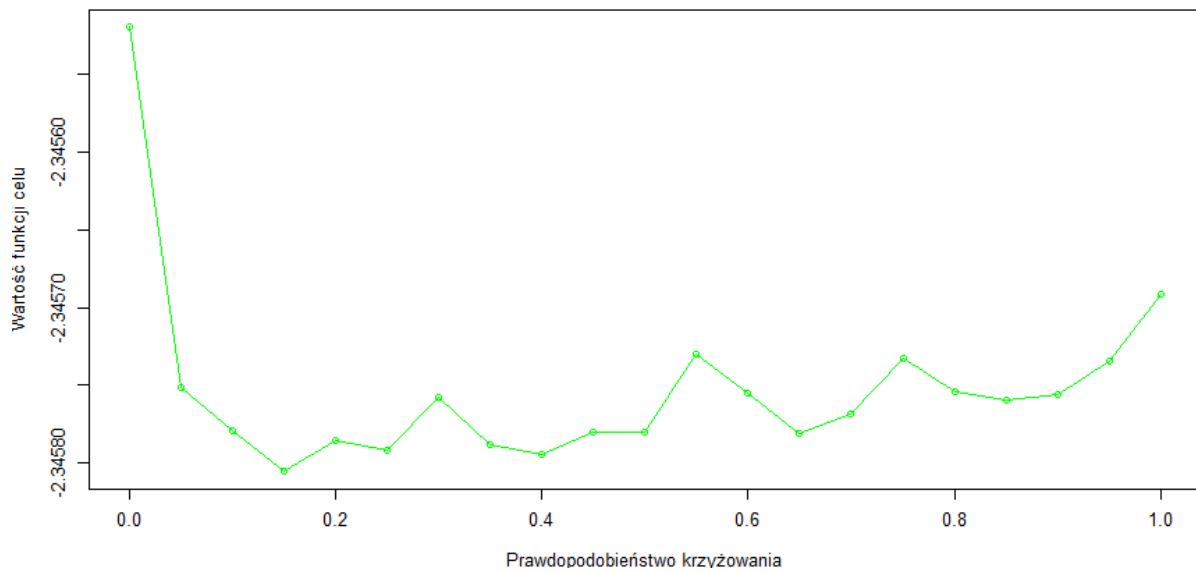


Rysunek 5.19 Powiększenie wykresu dla średniej z całej populacji

5.4.2 Badanie wpływu prawdopodobieństwa mutacji na przystosowanie dla funkcji Hosaki



Rysunek 5.20 Zależność wartości funkcji celu Hosaki dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od prawdopodobieństwa krzyżowania



Rysunek 5.21 Powiększenie wykresu dla najlepszego osobnika

5.4.3 Obserwacje nt. wpływu prawdopodobieństwa krzyżowania na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika

Na podstawie badań poczyniono następujące obserwacje:

- w przypadku funkcji Schuberta, zwiększenie prawdopodobieństwa do wartości ok. 0,8 powodowało obniżenie ogólnej jakości populacji, jednak do pewnej wartości (ok. 0,5) poprawie ulegało przystosowanie najlepszego osobnika,
- w przypadku funkcji Hosaki, wpływ ten nie był już tak jednoznaczny – dla średniej jakości całej populacji najkorzystniejsza była wartość ok. 0,9, natomiast ze względu na najlepszego osobnika – ok. 0,15.

5.5 BADANIE WPLYWU PRAWDOPODOBIENSTWA KRZYŻOWANIA I MUTACJI

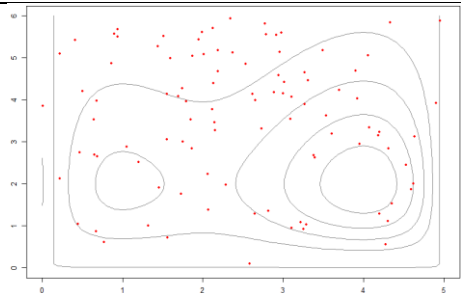
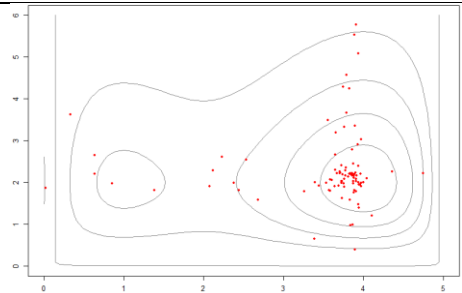
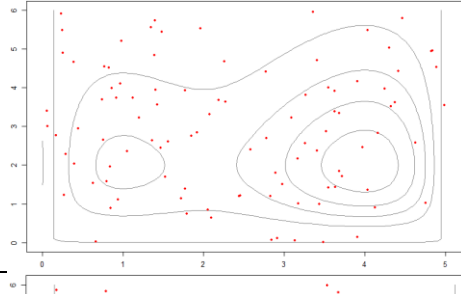
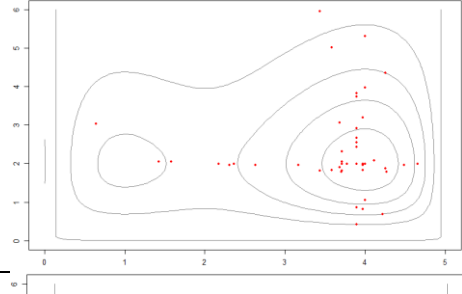
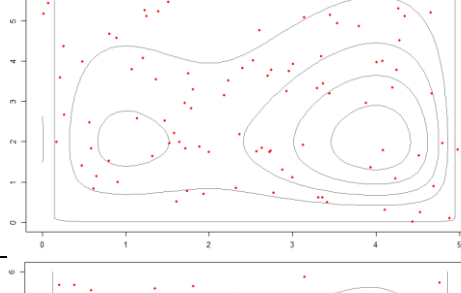
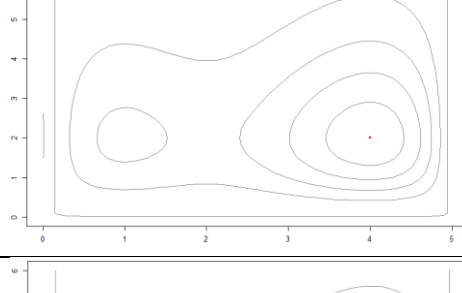
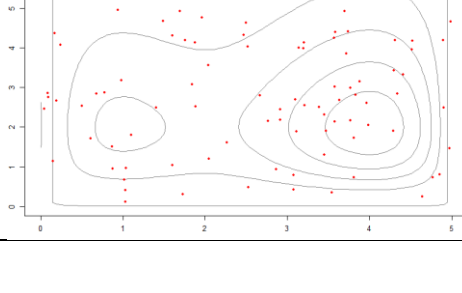
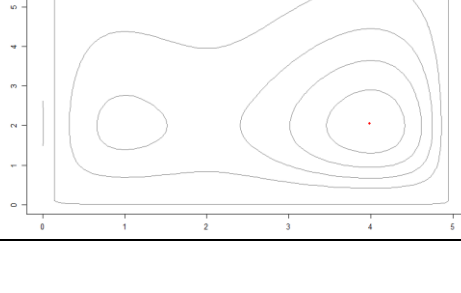
Badanie to ma nieco inny charakter od pozostałych ponieważ miało na celu sprawdzenie, jak zachowa się algorytm przy jednoczesnym wyłączeniu mutacji i krzyżowania, tzn. przy założeniu:

$$P_{\text{mutacji}} = 0 \wedge P_{\text{krzyżowania}} = 0$$

Dla zobrazowania przedstawiono różnice w ewolucji dla populacjach:

- gdzie zachodzi i krzyżowanie i mutacja,
- gdzie zachodzi tylko mutacja,
- gdzie zachodzi tylko krzyżowania,
- gdzie nie zachodzi ani krzyżowanie, ani mutacja.

Tabela 5.2 Poszczególne osobniki populacji w 1. i 60. pokoleniu w zależności od wartości prawdopodobieństw: mutacji i krzyżowania dla funkcji Hosaki

P_{mutacji}	$P_{\text{krzyżowania}}$	1. pokolenie	60. pokolenie
0,8	0,2		
0	0,2		
0,8	0		
0	0		

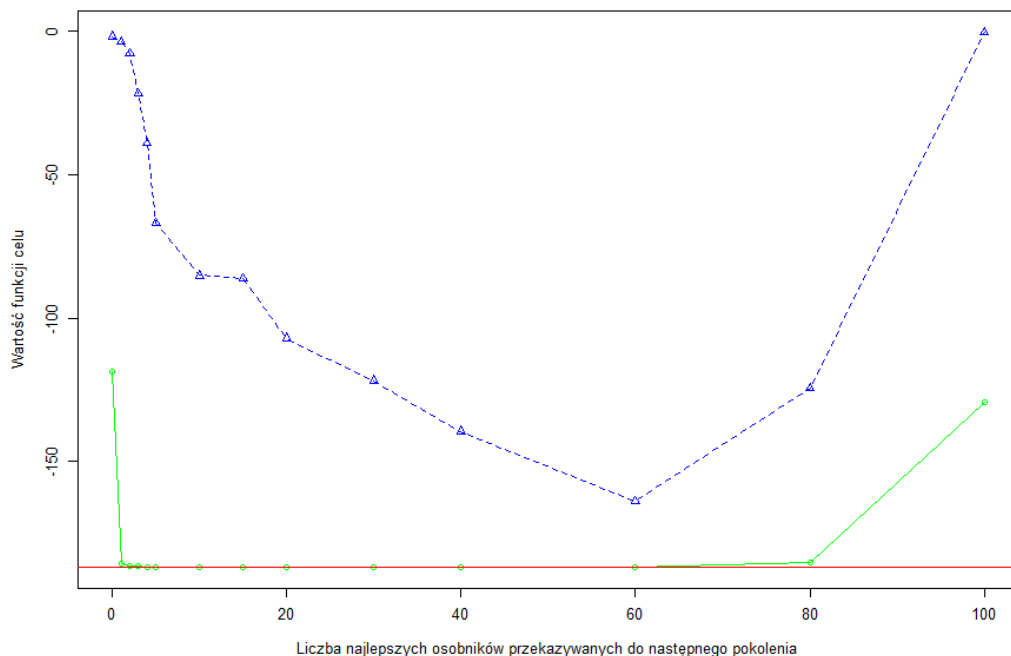
Z eksperymentu można wyciągnąć następujące wnioski:

- przy wyłączeniu operatora krzyżowania i mutacji, zadanie sprowadzało się do wyboru najlepszego spośród dostępnych (wylosowanych) osobników z populacji początkowej. W powyższej tabeli przedstawiono przykłady dla populacji 100 osobników – stąd liczba wylosowanych osobników jest dość duża, co pozwoliło na dość dokładny wybór najlepszego spośród nich.
- przy wyłączeniu krzyżowania i stosowaniu jedynie operatora mutacji zbieżność wartości funkcji celu w kolejnych pokoleniach jest mniejsza. Wynika to z tego, że zmiany w genotypie są przypadkowe. W populacji pojawiają się osobniki zmutowane o wartościach cech odbiegających od pozostałych,
- przy wyłączeniu mutacji i stosowaniu jedynie operatora krzyżowania maksimum globalne na powyższym przykładzie zostało odnalezione. Jednak ogólnie, brak mutacji grozi utknięciem w minimum lokalnym.

5.6 BADANIE WPŁYWU POZIOMU ELITARYZMU

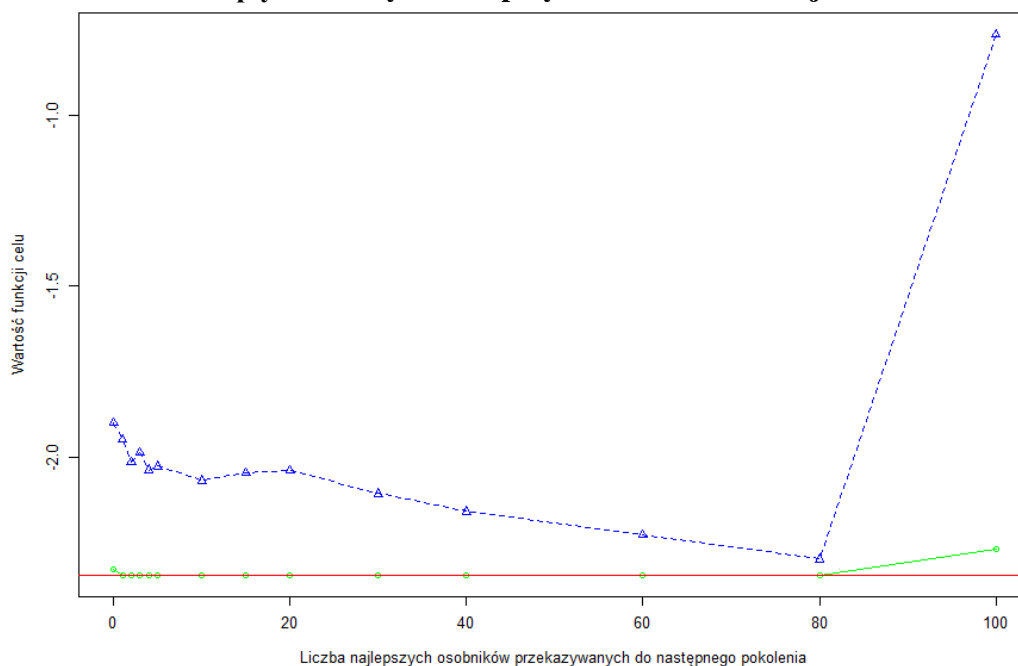
Elitaryzm określa ile najlepszych osobników z pokolenia zostanie przekazanych do następnego pokolenia bez zmian. W skrajnym przypadku może nie występować w ogóle, lub też obejmować całą populację (całe pokolenie).

5.6.1 Badanie wpływu elitaryzmu na przystosowanie do funkcji Schuberta

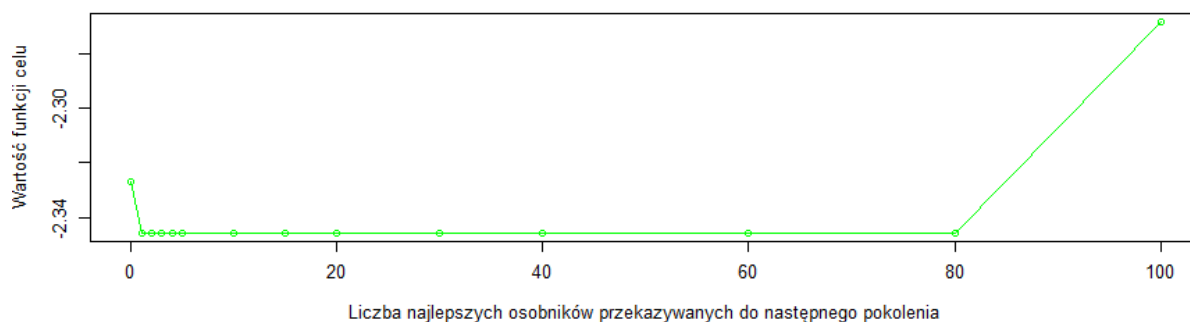


Rysunek 5.22 Zależność wartości funkcji celu Schuberta dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od wartości parametru „elitism”

5.6.2 Badanie wpływu elitaryzmu na przystosowanie do funkcji Hosaki



Rysunek 5.23 Zależność wartości funkcji celu Hosaki dla najlepszego osobnika z populacji oraz średniej wartości dla całej populacji w zależności od wartości parametru „elitism”



Rysunek 5.24 Powiększenie wykresu dla najlepszego osobnika.

5.6.3 Obserwacje nt. wpływu elitaryzmu na średnie przystosowanie populacji oraz przystosowanie najlepszego osobnika

Na podstawie badań poczyniono następujące obserwacje:

- zerowa wartość parametru „elitism” powoduje, że najlepszy osobnik z danego pokolenia jest „gubiony”, tzn. może nie zostać przekazany do następnego pokolenia,
- zwiększenie tego parametru już do wartości 1 zapobiega wyżej opisanemu zjawisku – stąd wartość najlepszego osobnika utrzymuje się na stałym poziomie w szerokim zakresie wartości parametru „elitism”,
- mimo, że wartość najlepszego osobnika jest już zachowywana od wartości parametru „elitism” równej 1, to dalsze zwiększanie udziału przekazywanych do następnego pokolenia najlepszych osobników z pokolenia poprzedniego pozytywnie wpływa na średnie przystosowanie populacji,
- jednak zbyt duża liczba przekazywanych osobników – równa liczebności całej populacji – powoduje ponowne, znaczące pogorszenie wartości funkcji celu – przekazywana jest bowiem cała populacja (wówczas określenie „najlepsze” w przypadku osobników z „elity” nie ma większego sensu, gdyż wszystkie osobniki przekazywane są do kolejnego pokolenia).

6 WNIOSKI

6.1 WNIOSKI NT. ALGORYTMU GENETYCZNEGO

Algorytmy ewolucyjne, w tym algorytm genetyczny mogą z powodzeniem być stosowane do zadań optymalizacji funkcji, których optymalizacja innymi algorytmami jest złożona obliczeniowo lub w ogóle nie możliwa - ze względu na skomplikowaną lub w ogóle nieznaną postać.

Przykładowo, obliczenie wartości funkcji dla wszystkich możliwych osobników (przegląd zupełny) – może w ogóle nie być możliwy. W takim przypadku zastosowanie algorytmu genetycznego pozwala operować tylko na pewnej, ograniczonej puli osobników, które poprzez zastosowanie odpowiednich operatorów genetycznych ewoluują w kolejnych pokoleniach optymalizując funkcję celu.

Algorytm genetyczny ma tę zaletę, że może zostać w dowolnej chwili przerywany – i wówczas zostanie zwrócona wartość najlepszego osobnika z danej populacji. Dla porównania algorytm zachłanny wymagałby wykonania w całości.

Algorytm genetyczny, choć z nazwy jest algorytmem jest w rzeczywistości metaheurystyką, na którą składają się heurystyki:

- selekcji osobników,
- krzyżowania osobników,
- mutacji osobników.

Skuteczność algorytmu genetycznego jest uzależniona od doboru parametrów oraz heurystyk. W tym ćwiczeniu badano jedynie wartości parametrów – nie zmieniano natomiast heurystyk (np. nie porównywano selekcji metodą turnieju z metodą ruletki).

Na podstawie doświadczeń wyciągnięto następujące wnioski:

- większa liczba osobników w populacji pozwala na szybsze znalezienie optimum globalnego, jednak przy stałej liczbie pokoleń zwiększenie liczebności populacji powoduje, że populacja nie zdąży wyewoluować do takiego poziomu jak populacja o mniejszej liczebności,
- elitaryzm poprawia jakość populacji i najlepszego osobnika tylko w ograniczonym zakresie. Brak elitaryzmu oznacza gubienie najlepszego osobnika z poprzedniego pokolenia. Liczność „elity” równa liczności populacji oznacza w praktyce kopiowanie całej populacji do następnego pokolenia,
- duże prawdopodobieństwo mutacji niekorzystnie wpływa na średnie przystosowanie całej populacji. Natomiast brak mutacji oznacza niebezpieczeństwo utknięcia w minimum lokalnym,
- wraz z rosnącą liczbą pokoleń, zarówno przystosowanie najlepszego osobnika jak i średnie przystosowanie całej populacji ulega poprawie, jednak tylko do pewnego momentu (pewnego pokolenia). Następnie wykres ulega stabilizacji i funkcja celu nie jest już dłużej optymalizowana. W przypadku najlepszego osobnika może oznaczać to znalezienie optimum.

6.2 WNIOSKI NT. IMPLEMENTACJI I JĘZYKA R

- język R stanowi bardzo ciekawą alternatywę do innych języków programowania,
- szczególnie przydatne są funkcyjne aspekty tego języka, które umożliwiły implementację skryptów bez zbędnego narzutu programistycznego (np. zwracanie funkcji jako wynik funkcji, która jako argument przyjmuje nazwę funkcji z pakietu `globalOptTest`),
- jeden z autorów sprawozdania miał okazję wcześniej implementować już podobne zadanie (z własną implementacją algorytmu genetycznego) w języku Kotlin. Ilość kodu potrzebna np. na przeprowadzenie badań (pomijając implementację samego algorytmu, której w tym przypadku nie było – skorzystano z gotowej paczki GA) w przypadku języka R jest znacząco niższa. Prostsze jest również generowanie wykresów,
- język R zawiera jednak kilka „puapek”. Jedną z nich jest problem, na który spowodował opóźnienie w oddaniu listy: podając nazwę elementu listy można bowiem używać składni *`nazwa.listy$nazwa.elementu`* lub *`nazwa.listy[[nazwa.elementu]]`*. Okazało się że jedynie ten drugi sposób umożliwia przekazanie nazwy elementu jako zmiennej typu string. Początkowo użyta składnia *`nazwa.listy$nazwa.elementu`* (gdzie *`nazwa.elementu`* to zmienna typu string) nie powodowała błędów, ale tworzyła w liście nowy element o nazwie „nazwa.elementu”. Stąd znalezienie tego błędu, znacząco wydłużyło czas implementacji.

7 KOD Z KOMENTARZEM

Komentarze umieszczono w kodzie w języku angielskim, dla zachowania spójności z nazewnictwem metod.

7.1 PRZEGLĄD NAJWAŻNIEJSZYCH FRAGMENTÓW KODU

Celem ułatwienia korzystania z pakietu *globalOptTests* napisano funkcję, która jako argument pobiera nazwę funkcji zdefiniowanej w pakiecie, i zwraca funkcję opakowującą funkcję pakietową *goTest*.

Funkcja ma więc sygnaturę $(string) \rightarrow ((\mathbb{R}, \mathbb{R}) \rightarrow \mathbb{R})$, gdzie $((\mathbb{R}, \mathbb{R}) \rightarrow \mathbb{R})$ odpowiada zwracanej funkcji celu. Jest to praktyczne zastosowanie funkcyjnych aspektów języka R.

Listing 7.1 Kod funkcji *objective.fun.of*

```
# returns objective function given by name
objective.fun.of <- function(function.name) {
  # Get the length of the parameter vector expected by a given objective function.
  problem.dimen <- getProblemDimen(function.name)

  function(x1, x2) {
    x1x2 <- c(x1, x2)
    goTest(par = c(x1x2, rep(0, problem.dimen - length(x1x2))), fnName = function.name,
    checkDim = TRUE)
  }
}
```

Do obliczenia wartości funkcji celu zadanej jej nazwą, utworzono funkcję *objective.fun.get*, która oblicza wartość funkcji celu dla wszystkich argumentów z zakresu określonego domyślnymi zakresami argumentów tejże funkcji celu. Funkcja została napisana dla 2-wymiarowych funkcji celu i zwraca listę $(x1, x2, values)$, gdzie $x1$ i $x2$ to wektory (spróbkowanych) wartości argumentu 1. i 2. (z krokiem 0,1), a $values$ to macierz wartości funkcji.

Listing 7.2 Kod funkcji *objective.fun.get*

```
objective.fun.get <- function(function.name) {
  # Default lower and upper bounds (box constraints) for the given objective function
  default.bounds <- getDefaultBounds(function.name)

  x1 <- seq(default.bounds$lower[1], default.bounds$upper[1], by = 0.1)
  x2 <- seq(default.bounds$lower[2], default.bounds$upper[2], by = 0.1)

  # Actual objective function of given name
  objective.fun <- objective.fun.of(function.name)

  # Apply objective.fun to each pair (x1', x2') where x1' belongs to x1 and x2 belongs to x2
  objective.fun.value <- outer(x1, x2, Vectorize(objective.fun))
  list(x1 = x1, x2 = x2, values = objective.fun.value)
}
```

Funkcja ta znalazła zastosowanie m.in. w funkcji *objective.fun.plot*, służącej do wyświetlania funkcji celu na ekranie w postaci wykresu 3-wymiarowego i tzw. mapy cieplnej.

Listing 7.3 Kod funkcji *objective.fun.plot*

```
# plots function given by name
objective.fun.plot <- function(function.name, axes = {axis(1); axis(2)}) {
  objective.fun <- objective.fun.get(function.name)
  # filled.contour(objective.fun$x1, objective.fun$x2, objective.fun$values,
  plot.axes={axis(1); axis(2); points(-0.8,-7.7, pch=3, cex=2, col="black", lwd=4);
  points(-7.7, -0.8, pch=3, cex=2, col="black", lwd=4)}, color.palette = jet.colors)
  filled.contour(objective.fun$x1, objective.fun$x2, objective.fun$values,
  plot.axes=axes, color.palette=jet.colors)
  persp3D(objective.fun$x1, objective.fun$x2, objective.fun$value, theta = 45, phi = 25,
  expand = 0.5, ticktype = "detailed", axes = TRUE)
}
```

Do powtórzenia przebiegu algorytmu genetycznego dla zadanych parametrów zdefiniowano funkcję *GA.run.iterations*, która zwraca listę w postaci (*best.mean*, *mean.mean*, *iterations.best.scores*), gdzie *best.mean* to średnia z najlepszych wyników w każdej iteracji, a *mean.mean* to średnia z średnich wyników dla całej populacji dla każdej iteracji.

Listing 7.4 Kod funkcji *GA.run.iterations*

```
# runs function genetic algorithm for function given by name iterations.count times
# params is list that contains params for genetic algorithm
GA.run.iterations <- function(function.name, iterations.count, params) {
  default.bounds <- getDefaultBounds(function.name)
  objective.fun <- objective.fun.of(function.name)
  iteration.mean.scores <- NULL # accumulator for mean results in each iteration
  iteration.best.scores <- NULL # accumulator for best results in each iteration
  for (iteration in 1:iterations.count) {
    GA <- ga(type = "real-valued",
      fitness = function(x) -objective.fun(x[1], x[2]),
      lower = c(default.bounds$lower[1], default.bounds$lower[2]),
      upper = c(default.bounds$upper[1], default.bounds$upper[2]),
      popSize = params$pop.size,
      maxiter = params$max.iter,
      elitism = params$elitism,
      pcrossover = params$pcrossover,
      pmutation = params$pmutation)
    iteration.best.scores[iteration] <- -GA@fitnessValue # objective.fun(GA@solution[,
    1], GA@solution[, 2])
    iteration.mean.scores[iteration] <- mean(apply(GA@population, 1, function(x1x2) {
    objective.fun(x1x2[1], x1x2[2]) }))
  }
  list(best.mean = mean(iteration.best.scores), mean.mean = mean(iteration.mean.scores),
  iteration.best.scores = iteration.best.scores)
}
```

Funkcję tę wykorzystano w funkcji *GA.run.experiment*, która jako argumenty przyjmuje: nazwę funkcji, liczbę iteracji pętli uśredniającej wyniki, listę parametrów domyślnych oraz nazwę i zakres wartości parametru, który będzie ulegał zmianie. Zwracana jest lista (*best.scores*, *mean.scores*), zawierająca wektory odpowiednio najlepszych i średnich wyników dla każdej z wartości zadanego parametru spośród zadanego zakresu zmienności parametru.

Listing 7.5 Kod funkcji *GA.run.experiment*

```
# runs experiment for every param.name value in param.range
GA.run.experiment <- function(function.name, iterations.count, params, param.name,
  param.range) {
  param.best.scores <- NULL # accumulator for best mean results in generation
  param.mean.scores <- NULL # accumulator for mean mean results in generation
```



```

# GA.best <- NULL # best result
for (param.value.index in seq_along(param.range)) {
  params[[param.name]] <- param.range[param.value.index]
  param.score <- GA.run.iterations(function.name, iterations.count, params)
  param.best.scores[param.value.index] <- param.score$best.mean
  param.mean.scores[param.value.index] <- param.score$mean.mean
}
list(best.scores = param.best.scores, mean.scores = param.mean.scores)
}

```

Żeby całkowicie zautomatyzować przeprowadzenie badań zdefiniowano funkcję *GA.run.experiment.list*, która wykonuje zadane eksperymenty w trybie wsadowym, drukując dla każdego z nich odpowiednie wykresy na ekranie.

Listing 7.6 Kod funkcji *GA.run.experiment.list*

```

# runs experiments as a batch
GA.run.experiment.list <- function (function.name, param.names.and.labels,
param.value.ranges, iterations.count=10) {
  for (param.index in seq_along(param.names.and.labels$names)) {
    param.name <- param.names.and.labels$names[param.index]
    param.value.range <- param.value.ranges[[param.name]]
    param.score.range <- GA.run.experiment(function.name, iterations.count, params,
param.name, param.value.range)
    scores.plot(
      x = param.value.range,
      y1 = param.score.range$best.scores,
      y2 = param.score.range$mean.scores,
      const = getGlobalOpt(function.name),
      x.label = param.names.and.ranges$labels[param.index]
    )
  }
}

```

Wywołanie tej funkcji celem przeprowadzenia pełnego zestawu badań dla funkcji Schuberta przedstawiono w poniższym listingu.

Listing 7.7 Wywołanie funkcji *GA.run.experiment*

```

# experiments for Schuber function
function.name <- "Schubert"
params <- list(pop.size = 100, max.iter = 100, elitism = 0, pcrossover = 0.8, pmutation = 0.2)
param.names.and.labels <- list(
  names = c("pop.size", "max.iter", "elitism", "pcrossover", "pmutation"),
  labels = c("Rozmiar populacji", "Liczba pokoleń", "Poziom elityzmu",
"Prawdopodobieństwo krzyżowania", "Prawdopodobieństwo mutacji")
)
param.value.ranges <- list(
  pop.size = c(1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 60, 80, 100, 140, 160),
  max.iter = c(1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 60, 80, 100, 140, 160),
  elitism = c(0, 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 60, 80, 100),
  pcrossover = seq(0, 1, by = 0.1),
  pmutation = seq(0, 1, by = 0.1)
)
GA.run.experiment.list(function.name, param.names.and.labels, param.value.ranges)

```

Do pojedynczego wywołania algorytmu genetycznego dla funkcji celu zadanej jej nazwą oraz dla zadanej listy parametrów i argumentu opcjonalnego – funkcji monitorującej przebieg wykonania algorytmu, napisano funkcję *GA.run.once*, zwracającą obiekt *GA* [4]. Oprócz wartości zwracanej,

funkcja może powodować efekt uboczny w przypadku przekazania funkcji monitorującej – np. drukowanie osobników z kolejnych pokoleń na ekranie monitora.

Listing 7.8 Funkcja GA.run.once

```
# runs GA only once for function given by name with params
# monitor function can be attached to show progress in each generation
GA.run.once <- function(function.name, params, monitor=NULL) {
  default.bounds <- getDefaultBounds(function.name)
  objective.fun <- objective.fun.of(function.name)
  ga(type = "real-valued",
     fitness = function(x) -objective.fun(x[1], x[2]),
     lower = c(default.bounds$lower[1], default.bounds$lower[2]),
     upper = c(default.bounds$upper[1], default.bounds$upper[2]),
     popSize = params$pop.size,
     maxiter = params$max.iter,
     elitism = params$elitism,
     pcrossover = params$pcrossover,
     pmutation = params$pmutation,
     monitor = monitor
  )
}
```

7.2 PEŁNY KOD PROGRAMU

Listing 7.9 Skrypt global_opt.R

```
# dependencies
require("GA")
require("globalOptTests")

function.names <- c("Schubert", "Hosaki")

# returns objective function given by name
objective.fun.of <- function(function.name) {
  # Get the length of the parameter vector expected by a given objective function.
  problem.dimen <- getProblemDimen(function.name)

  function(x1, x2) {
    x1x2 <- c(x1, x2)
    goTest(par = c(x1x2, rep(0, problem.dimen - length(x1x2))), fnName = function.name,
    checkDim = TRUE)
  }
}

objective.fun.get <- function(function.name) {
  # Default lower and upper bounds (box constraints) for the given objective function
  default.bounds <- getDefaultBounds(function.name)

  x1 <- seq(default.bounds$lower[1], default.bounds$upper[1], by = 0.1)
  x2 <- seq(default.bounds$lower[2], default.bounds$upper[2], by = 0.1)

  # Actual objective function of given name
  objective.fun <- objective.fun.of(function.name)

  # Apply objective.fun to each pair (x1', x2') where x1' belongs to x1 and x2 belongs to x2
  objective.fun.value <- outer(x1, x2, Vectorize(objective.fun))
  list(x1 = x1, x2 = x2, values = objective.fun.value)
}

# plots function given by name
```

```

objective.fun.plot <- function(function.name, axes = {axis(1); axis(2)}) {
  objective.fun <- objective.fun.get(function.name)
  # filled.contour(objective.fun$x1, objective.fun$x2, objective.fun$values,
  plot.axes={axis(1); axis(2); points(-0.8,-7.7, pch=3, cex=2, col="black", lwd=4);
  points(-7.7, -0.8, pch=3, cex=2, col="black", lwd=4)}, color.palette = jet.colors)
  filled.contour(objective.fun$x1, objective.fun$x2, objective.fun$values,
  plot.axes=axes, color.palette=jet.colors)
  persp3D(objective.fun$x1, objective.fun$x2, objective.fun$value, theta = 45, phi = 25,
  expand = 0.5, ticktype = "detailed", axes = TRUE)
}

# objective.fun.plot(function.names[1])

# finds global optimums of function given by name
objective.fun.opt.indicies <- function(function.name) {
  objective.fun.value <- (objective.fun.get(function.name))$values
  which(objective.fun.value == min(objective.fun.value), arr.ind = TRUE)
}

# translates indicies into values for function given by name
objective.fun.opt.indicies.translate <- function(matrix.of.indicies, function.name) {
  # Default lower and upper bounds (box constraints) for the given objective function
  default.bounds <- getDefaultBounds(function.name)

  x1 <- seq(default.bounds$lower[1], default.bounds$upper[1], by = 0.1)
  x2 <- seq(default.bounds$lower[2], default.bounds$upper[2], by = 0.1)
  result <- NULL
  add.to.result <- function(row.col) {
    append(result, c(x1[row.col[1]], x2[row.col[2]]))
  }
  apply(matrix.of.indicies, 1, add.to.result)
  result
}

# translates indicies into values for function given by name
objective.fun.opt.indicies.translate.one <- function(indicies, function.name) {
  # Default lower and upper bounds (box constraints) for the given objective function
  default.bounds <- getDefaultBounds(function.name)

  x1 <- seq(default.bounds$lower[1], default.bounds$upper[1], by = 0.1)
  x2 <- seq(default.bounds$lower[2], default.bounds$upper[2], by = 0.1)
  c(x1[indicies[1]], x2[indicies[2]])
}

```

Listing 7.10 Skrypt *global_opt.R*

```
GA.run.iterations <- function(function.name, iterations.count, params) {
  default.bounds <- getDefaultBounds(function.name)
  objective.fun <- objective.fun.of(function.name)
  iteration.mean.scores <- NULL # accumulator for mean results in each iteration
  iteration.best.scores <- NULL # accumulator for best results in each iteration
  for (iteration in 1:iterations.count) {
    GA <- ga(type = "real-valued",
             fitness = function(x) -objective.fun(x[1], x[2]),
             lower = c(default.bounds$lower[1], default.bounds$lower[2]),
             upper = c(default.bounds$upper[1], default.bounds$upper[2]),
             popSize = params$pop.size,
             maxiter = params$max.iter,
             elitism = params$elitism,
             pcrossover = params$pcrossover,
             pmutation = params$pmutation)
    iteration.best.scores[iteration] <- -GA@fitnessValue # objective.fun(GA@solution[,
1], GA@solution[, 2])
    iteration.mean.scores[iteration] <- mean(apply(GA@population, 1, function(x1x2) {
objective.fun(x1x2[1], x1x2[2]) }))
  }
  list(best.mean = mean(iteration.best.scores), mean.mean = mean(iteration.mean.scores),
iteration.best.scores = iteration.best.scores)
}

# runs experiment for every param.name value in param.range
GA.run.experiment <- function(function.name, iterations.count, params, param.name,
param.range) {
  param.best.scores <- NULL # accumulator for best mean results in generation
  param.mean.scores <- NULL # accumulator for mean mean results in generation
  # GA.best <- NULL # best result
  for (param.value.index in seq_along(param.range)) {
    params[[param.name]] <- param.range[param.value.index]
    param.score <- GA.run.iterations(function.name, iterations.count, params)
    param.best.scores[param.value.index] <- param.score$best.mean
    param.mean.scores[param.value.index] <- param.score$mean.mean
  }
  list(best.scores = param.best.scores, mean.scores = param.mean.scores)
}

# runs experiments as a batch
GA.run.experiment.list <- function(function.name, param.names.and.labels,
param.value.ranges, iterations.count=10) {
  for (param.index in seq_along(param.names.and.labels$names)) {
    param.name <- param.names.and.labels$names[param.index]
    param.value.range <- param.value.ranges[[param.name]]
    param.score.range <- GA.run.experiment(function.name, iterations.count, params,
param.name, param.value.range)
    scores.plot(
      x = param.value.range,
      y1 = param.score.range$best.scores,
      y2 = param.score.range$mean.scores,
      const = getGlobalOpt(function.name),
      x.label = param.names.and.ranges$labels[param.index]
    )
  }
}

#plots best (y1) and mean (y2) scores for each param value x
scores.plot <- function(x, y1, y2, const, x.label, y.label = "Wartość funkcji celu") {
  matplot(x, cbind(y1, y2), col = c("green", "blue"), type = c("o", "o"), pch = 1:2, xlab
= x.label, ylab = y.label)
  abline(h = const, col = "red")
}
```

```

}

# experiments for Schubert function
function.name <- "Schubert"
params <- list(pop.size = 100, max.iter = 100, elitism = 0, pcrossover = 0.8, pmutation = 0.2)
param.names.and.labels <- list(
  names = c("pop.size", "max.iter", "elitism", "pcrossover", "pmutation"),
  labels = c("Rozmiar populacji", "Liczba pokoleń", "Poziom elityzmu",
"Prawdopodobieństwo krzyżowania", "Prawdopodobieństwo mutacji")
)
param.value.ranges <- list(
  pop.size = c(1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 60, 80, 100, 140, 160),
  max.iter = c(1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 60, 80, 100, 140, 160),
  elitism = c(0, 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 60, 80, 100),
  pcrossover = seq(0, 1, by = 0.1),
  pmutation = seq(0, 1, by = 0.1)
)

GA.run.experiment.list(function.name, param.names.and.labels, param.value.ranges)

```

Listing 7.11 Skrypt main.R

```

# plots each generation of GA
monitor <- function(obj)
{
  contour(objective$x1, objective$x2, objective$values, drawlabels = FALSE, col =
grey(0.5))
  title(paste("iteration =", obj@iter), font.main = 1)
  points(obj@population, pch = 20, col = 2)
  Sys.sleep(0.2)
}

# runs GA only once for function given by name with params
# monitor function can be attached to show progress in each generation
GA.run.once <- function(function.name, params, monitor=NULL) {
  default.bounds <- getDefaultBounds(function.name)
  objective.fun <- objective.fun.of(function.name)
  ga(type = "real-valued",
    fitness = function(x) -objective.fun(x[1], x[2]),
    lower = c(default.bounds$lower[1], default.bounds$lower[2]),
    upper = c(default.bounds$upper[1], default.bounds$upper[2]),
    popSize = params$pop.size,
    maxiter = params$max.iter,
    elitism = params$elitism,
    pcrossover = params$pcrossover,
    pmutation = params$pmutation,
    monitor = monitor
  )
}

params <- list(pop.size = 100, max.iter = 100, elitism = 1, pcrossover = 0.01, pmutation
= 0.0)
function.name <- "Hosaki"

ga.result <- GA.run.once(function.name, params, monitor)

```

8 LITERATURA

- [1] http://infinity77.net/global_optimization/test_functions_nd_H.html
- [2] <https://www.sfu.ca/~ssurjano/shubert.html>
- [3] https://pl.wikipedia.org/wiki/Algorytm_genetyczny
- [4] <https://cran.r-project.org/web/packages/GA/vignettes/GA.html>