Inteligencja obliczeniowa i jej zastosowania

Laboratorum cz. II, nr 3-5

Autorzy:
Agnieszka Wątrucka, nr indeksu: 200016
Joanna Piątek, nr indeksu: 199966

Grupa: Środa, 15:15

6 maja 2017

Prowadzący: prof. dr hab. inż. Olgierd Unold

1 Własne funkcje mutacji, krzyżowania i selekcji

1.1 Wstęp

Zadanie polegało na zastąpieniu domyślnych funkcji używanych w pakiecie GA na własne implementacje i porównanie ich działania. Podstawowe zestawienie składa się z najlepszych oraz średnich wyników dla danej populacji w przypadku użycia funkcji wbudowanej oraz własnej. Podczas badań zmieniane są parametry dotyczące badanej funkcji, natomiast pozostałe przyjmują wartości domyślne podane poniżej.

Wartości domyślne funkcji użytych w badaniach to kolejno:
Rozmiar populacji - 100
Liczba iteracji – 50
Prawdopodobieństwo krzyżowania – 0.5
Prawdopodobieństwo wystąpienia mutacji - 0.1
Selekcja elitarnych jednostek – 6

Wyniki wszystkich badań to rezultaty uśrednione po 30 przebiegach.

1.2 Funkcja *branin*

Do badań została wykorzystana funkcja wielomodalna *branin*.

Wzór funkcji *branin*:

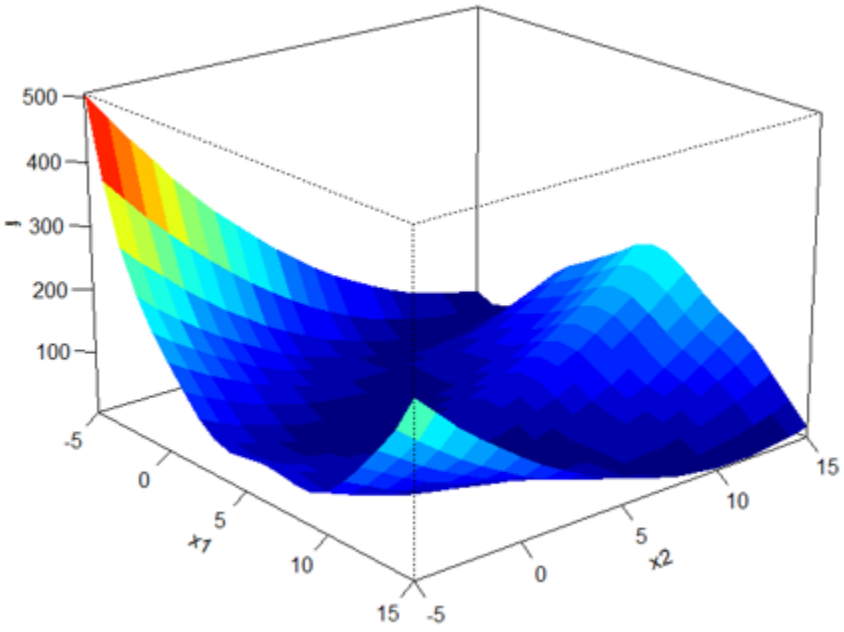
$$f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s$$

Minimum globalne:

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

Poniżej przedstawiono kod źródłowy importujący funkcję *branin* z pakietu *globalOptTests*.

```
library(globalOptTests)

branin <- function(x1,x2)
{
    return(goTest(fnName="Branin", par=c(x1,x2)))
}
```



Rysunek 1: Wykres funkcji *branin*

1.3 Główny kod źródłowy

Poniższe kody źródłowe zawierają definiowanie używanych funkcji uruchamiających algorytmy genetyczne z pakietu *GA*, a następnie uśredniających wyniki ich działania. Pierwszy fragment zawiera implementację z domyślnymi funkcjami mutacji, selekcji i krzyżowania, natomiast w następnym używane są funkcje autorskie. Na trzecim listingu przedstawione jest finalne wywołanie tych funkcji.
Warto zaznaczyć, że wynikiem optymalizacji są liczby przeciwne do tych, które powinno się otrzymać. Wynika to z następującej linii kodu w wywołaniu algorytmu *GA*:

```
fitness = function(x) - branin(x[1], x[2])
```

- funkcja podawana jest ze znakiem '-'. W celu odwrócenia znaku wyników otrzymane uśrednione wartości mnożymy przez -1, co można zobaczyć w trzecim listingu.

```
meanRowsMax <- -rowMeans(gaResult$max)
```

```
# Implementacja funkcji pozwalajacej na zmiane parametrow
meanGA <- function(population,iteration,crosing,mutant,mini,maxi,elit)
{
   xMax <- matrix ( 0 ,iteration,loopRepetitions)
   xMean <- matrix ( 0 ,iteration,loopRepetitions)

   # Petla do usredniania wynikow
   for ( i in 1:loopRepetitions )
   {
        GA <- ga(type = "real-valued",
        fitness = function(x) - branin(x[1], x[2]),
        min = mini, max = maxi, pcrossover = crosing, pmutation = mutant,
        popSize = population, maxiter = iteration, elitism = elit)

        xMax[,i] <- GA@summary[,1]
        xMean[,i] <- GA@summary[,2]
        print(GA@solution)
   }
   return(list(max=xMax,min=xMean))
}
```

```
# Wersja uzywajaca wlasnych implementacji funkcji (np. mutacji)
meanGaCustom <- function(population,iteration,crosing,mutant,mini,maxi,elit)
{
   xMax <- matrix ( 0 ,iteration,loopRepetitions)
   xMean <- matrix ( 0 ,iteration,loopRepetitions)

   # Petla do usredniania wynikow
   for ( i in 1:loopRepetitions )
   {
        GA <- ga(type = "real-valued",
        fitness = function(x) - branin(x[1], x[2]),

        # Wlasne implementacje funkcji odkomentowywane w miare potrzeb
        #mutation = myMutation,
        #selection = mySelection,
        crossover = myCrossover,

        min = mini, max = maxi, pcrossover = crosing, pmutation = mutant,
        popSize = population, maxiter = iteration, elitism = elit)

        xMax[,i] <- GA@summary[,1]
        xMean[,i] <- GA@summary[,2]
        print(GA@solution)
   }
   return(list(max=xMax,min=xMean,sol=GA@solution))
}
```

```
# Ustawienie parametrow
population <- 100
iteration <- 50
crosing <- 1
mutant <- 0.1
mini <- c(-5,-5)
maxi <- c(15,15)
elit <- 6

# Finalne wywolanie funkcji domyslnej
gaResult <- meanGA(population, iteration, crosing, mutant,
mini, maxi, elit)
# Wartosci srednie i najlepsze do wykresow
meanRowsMax <- -rowMeans(gaResult$max)
meanRowsMean <- -rowMeans(gaResult$min)

# Finalne wywolanie funkcji z wlasnymi implementacjami
gaResultCustom <- meanGaCustom(population, iteration, crosing, mutant,
mini, maxi, elit)
# Wartosci srednie i najlepsze do wykresow
meanRowsMaxCustom <- -rowMeans(gaResultCustom$max)
meanRowsMeanCustom <- -rowMeans(gaResultCustom$min)

# Pobranie sredniej wartosci dla ostatnich iteracji - wyniki koncowe
round(tail(meanRowsMean, n=1), digit=6)
round(tail(meanRowsMax, n=1), digit=6)
round(tail(meanRowsMeanCustom, n=1), digit=6)
round(tail(meanRowsMaxCustom, n=1), digit=6)
```

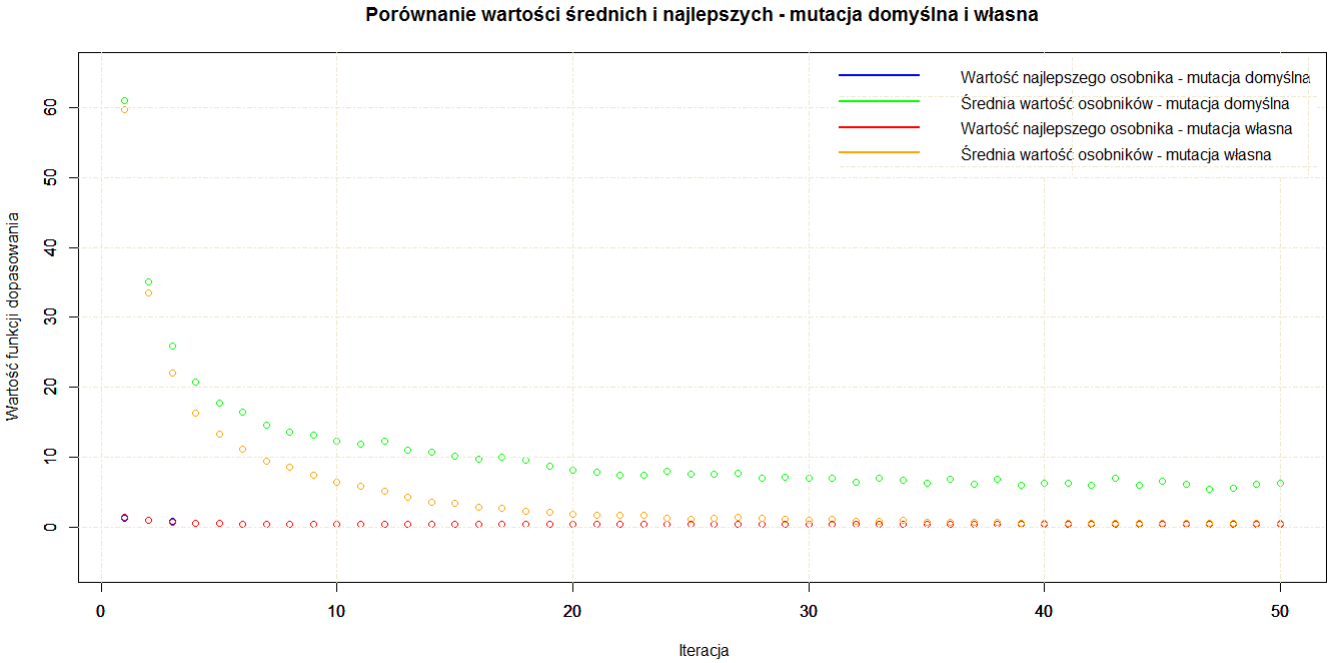
1.4 Mutacja

Zadaniem mutacji jest wprowadzenie różnorodności w populacji. Dlatego we własnej implementacji zostały wprowadzone elementy losowości.

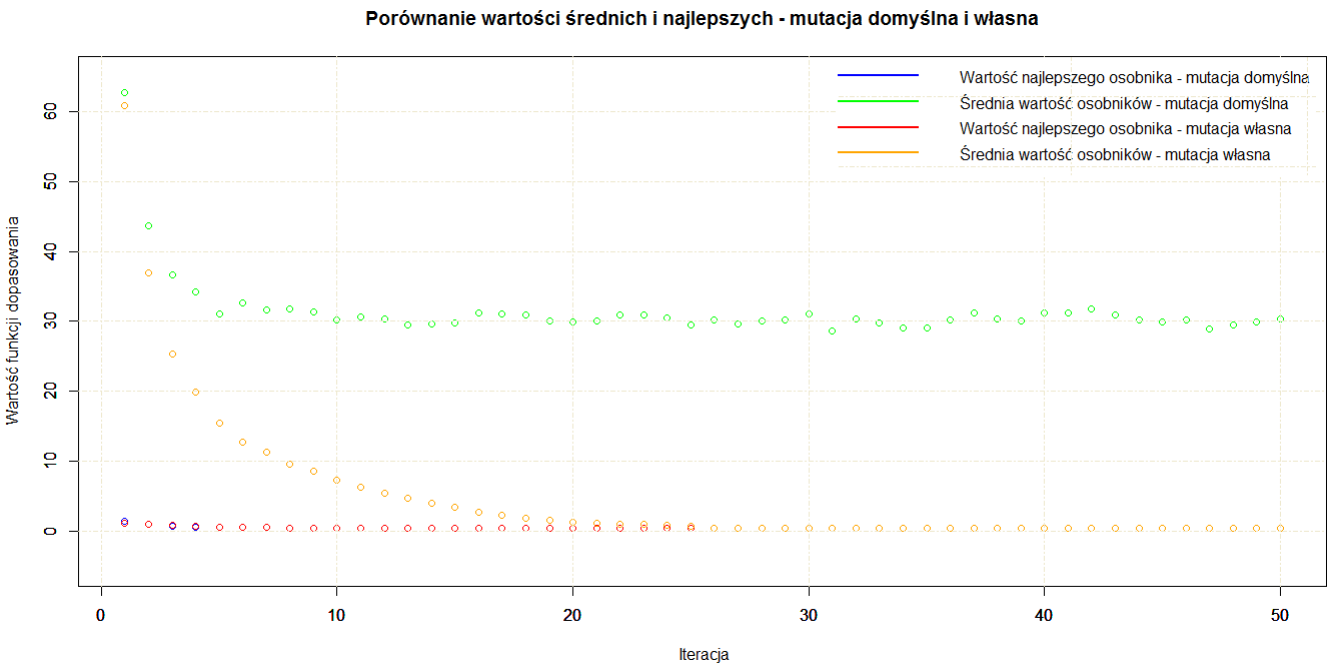
1.4.1 Kod źródłowy

```
myMutation <- function (ga_object, parent)
{
  # Pobranie osobnikow do mutacji
  mutate <- parent <- as.vector(ga_object@population[parent,])
  parLen <- length(parent)

  all <- c(1:parLen)
  # Wykluczenie losowych osobnikow z mutacji
  excluded <- sample(1:parLen/4, size = 1)
  indexesToMutate <- all[!excluded]

  # Mutacja przy uzyciu losowych wartosci
  mutate[indexesToMutate] <- mutate[indexesToMutate] - rnorm(n=1, m=0, sd=3)

  return(mutate)
}
```

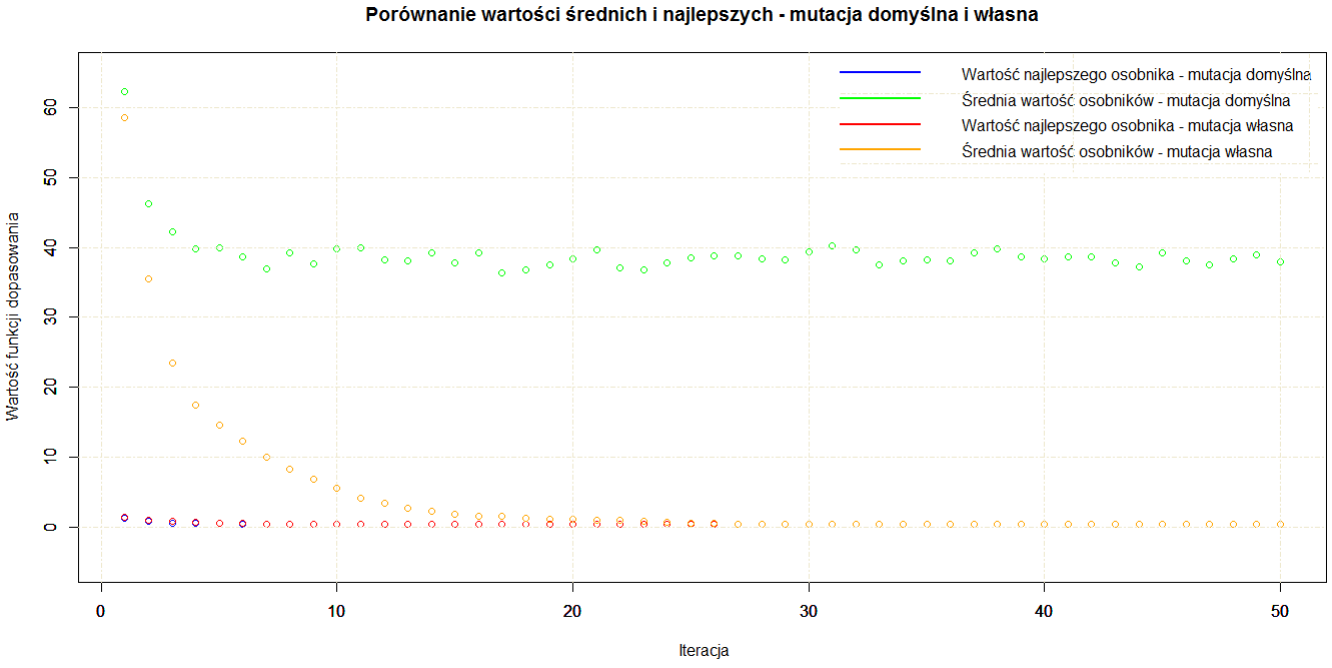
1.4.2 Wyniki badań



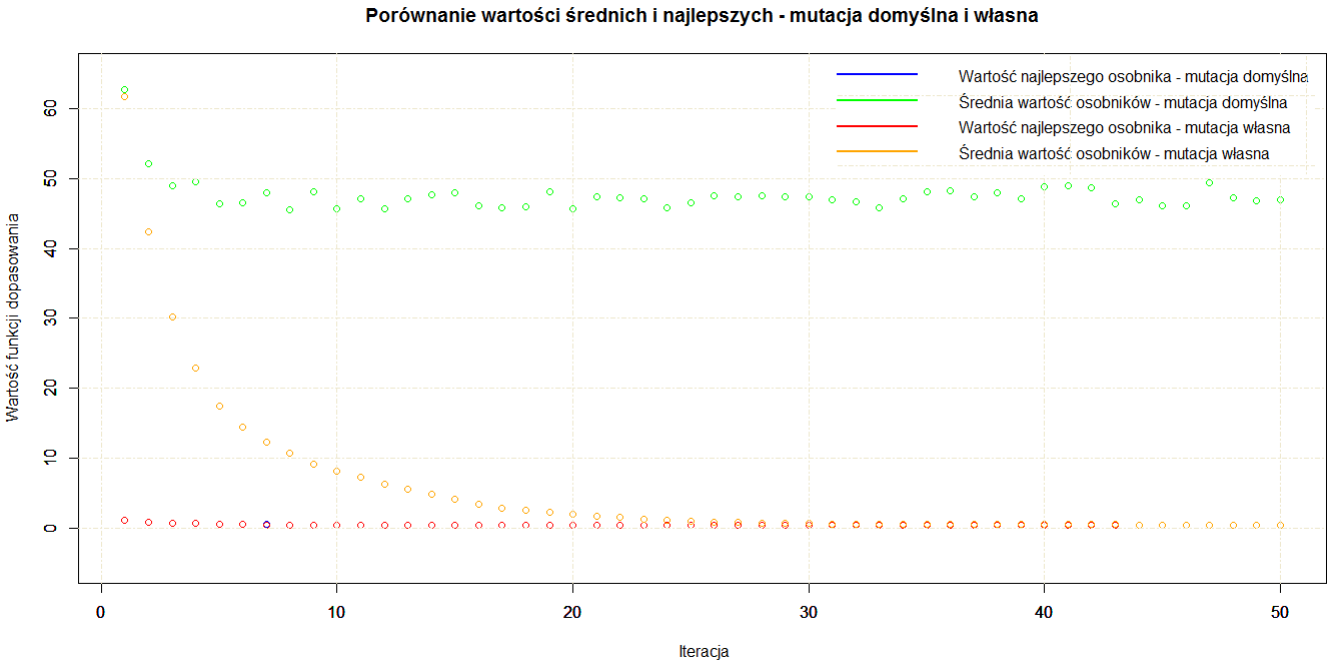
Rysunek 2: Wykres dla prawdopodobieństwa mutacji 0.1



Rysunek 3: Wykres dla prawdopodobieństwa mutacji 0.5



Rysunek 4: Wykres dla prawdopodobieństwa mutacji 0.7



Rysunek 5: Wykres dla prawdopodobieństwa mutacji 1

1.4.3 Wnioski

Tabela 1: Wartości średnie i najlepsze osobnika dla domyślnej i własnej funkcji mutacji

Prawdopodobieństwo mutacji	**Mutacja domyślna**		**Mutacja własna**	
	Wartość średnia	Najlepszy wynik	Wartość średnia	Najlepszy wynik
0.1	5.753710	**0.398006**	0.398736	0.398687
0.5	31.029570	0.401904	0.415733	0.398201
0.7	38.184020	0.404532	0.567679	0.398926
1	46.920430	0.408154	0.452637	0.400847

Na wykresach można łatwo zauważyć, że wybór funkcji mutacji nie wpłynął znacząco na wartości najlepszego osobnika w populacji, natomiast wartości średnie wyraźnie odstają od siebie dla tych dwóch przypadków. Można stąd wywnioskować, że własna implementacja mutacji wykonuje mniej inwazyjne operacje na osobnikach.
Natomiast najlepszy wynik przy tej ilości iteracji został osiągnięty podczas uruchomienia algorytmu z ustawieniami domyślnymi. Implementacja mutacji nie poprawiła więc wyników działania algorytmu.

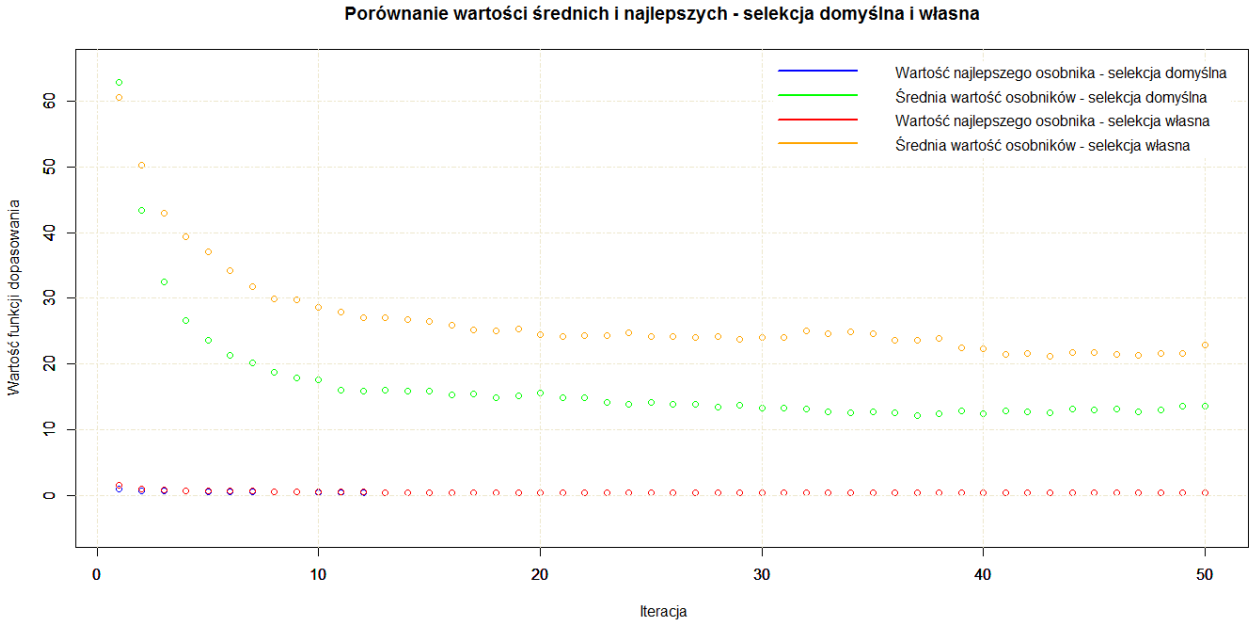
1.5 Selekcja elitarna

Za pomocą selekcji elitarnej wybierane są osobniki, które pojawią się także w następnym pokoleniu.

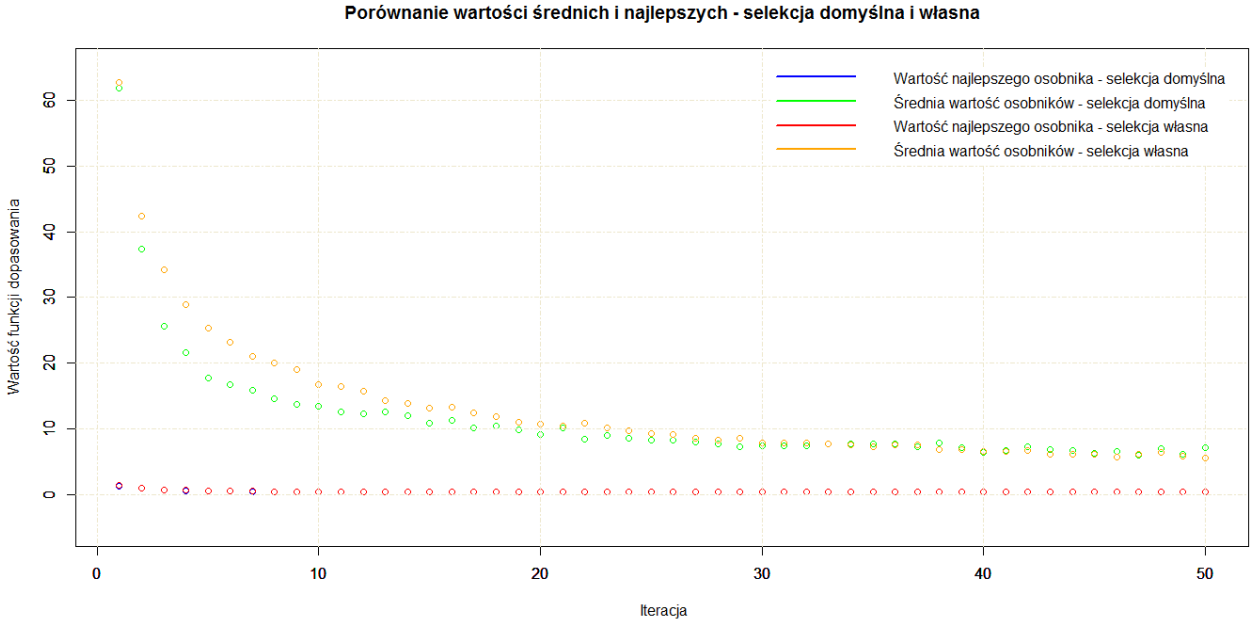
1.5.1 Kod źródłowy

```
mySelection <- function (ga_object, parent)
{
   # Pobranie specjalnie wyliczonego wspolczynnika
   # o rozmiarze rownym rozmiarowi populacji
   prob <- abs(
        cos(ga_object@fitness)/(sin(ga_object@fitness))
   )

   # Wektor o tym samym rozmiarze;
   # Losowo pobrane indeksy
   # mniejsze lub rowne rozmiarowi populacji
   sel <- sample(1:ga_object@popSize, size = ga_object@popSize,
        prob = pmin(prob, 1, na.rm = TRUE),
        replace = TRUE)

   # Selekcja na podstawie wektora sel
   out <- list(population = ga_object@population[sel,,drop=FALSE],
        fitness = ga_object@fitness[sel])
   return(out)
}
```

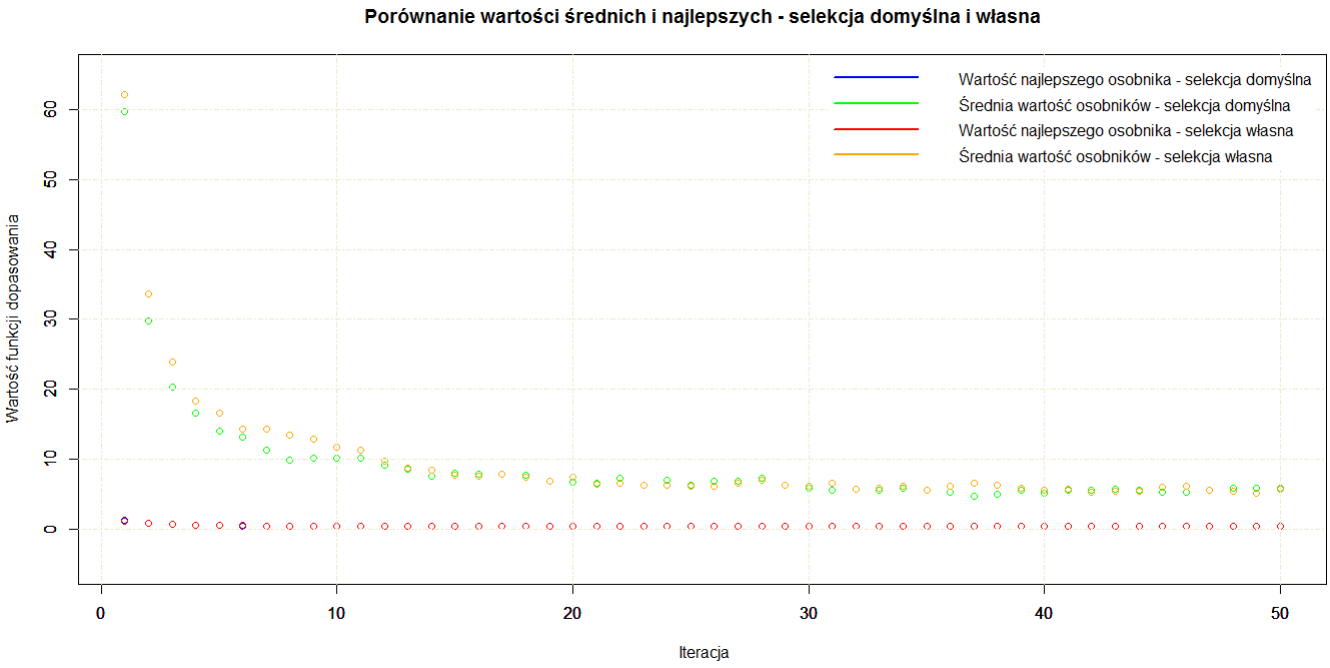
1.5.2 Wyniki badań



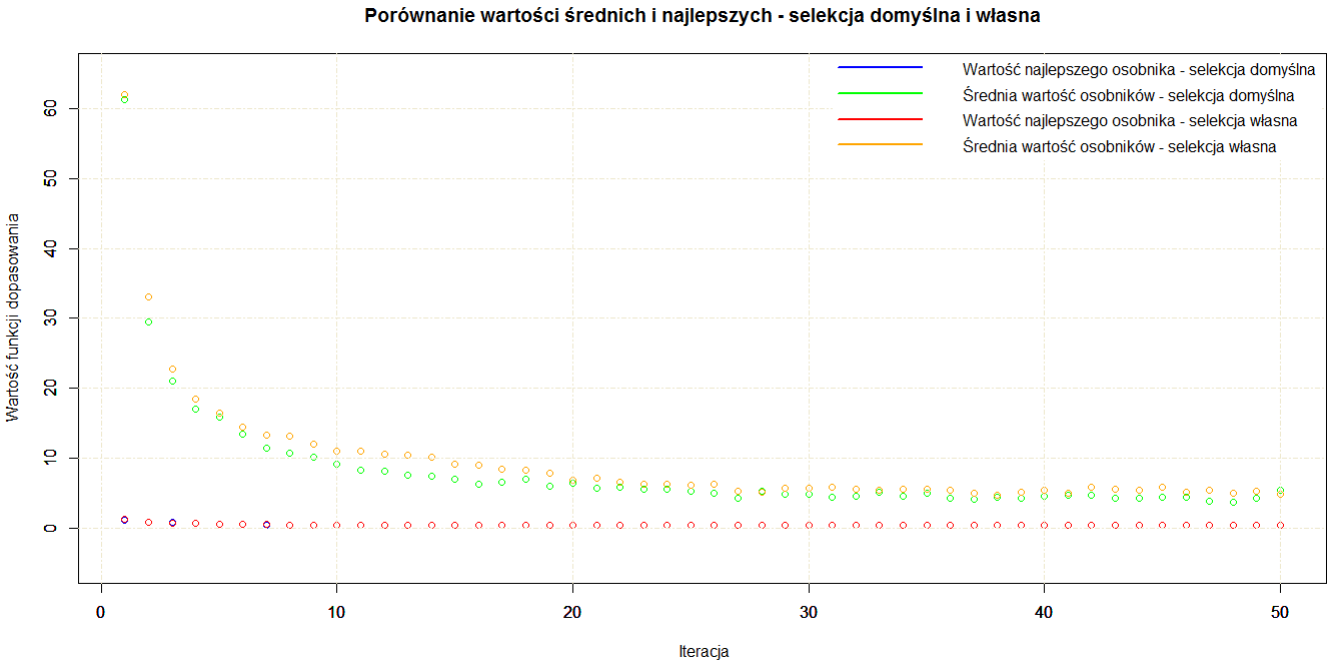
Rysunek 6: Wykres przy 1 osobniku elitarnym



Rysunek 7: Wykres przy 6 osobnikach elitarnych



Rysunek 8: Wykres przy 20 osobnikach elitarnych



Rysunek 9: Wykres przy 50 osobnikach elitarnych

1.5.3 Wnioski

Tabela 2: Wartości średnie i najlepsze osobnika dla domyślnej i własnej funkcji selekcji

Selekcja elitarna	**Selekcja domyślna**		**Selekcja własna**	
	Wartość średnia	Najlepszy wynik	Wartość średnia	Najlepszy wynik
1	13.660880	0.405495	22.860800	0.414707
6	7.106856	0.400895	5.544652	0.397909
20	5.847374	**0.397888**	5.678752	0.397977
50	5.42606	0.397903	4.833812	**0.397888**

Najlepsze wyniki zarówno dla selekcji domyślnej, jak i własnej, pojawiają się przy większych ilościach osobników elitarnych (różnią się o 1 na szóstej pozycji po przecinku od minimum globalnego). Można zauważyć, że nowa funkcja selekcji działa nieco inaczej i najlepsze rezultaty daje przy liczbie osobników elitarnych równej aż połowie populacji.
Natomiast dla 1 osobnika elitarnego przy użyciu zaimplementowanej funkcji wartości średnie osobników znaczie odstają od tych uzyskanych za pomocą selekcji pochodzącej z pakietu *GA*, także najlepsze wyniki nie są zbliżone do minimum lokalnego.

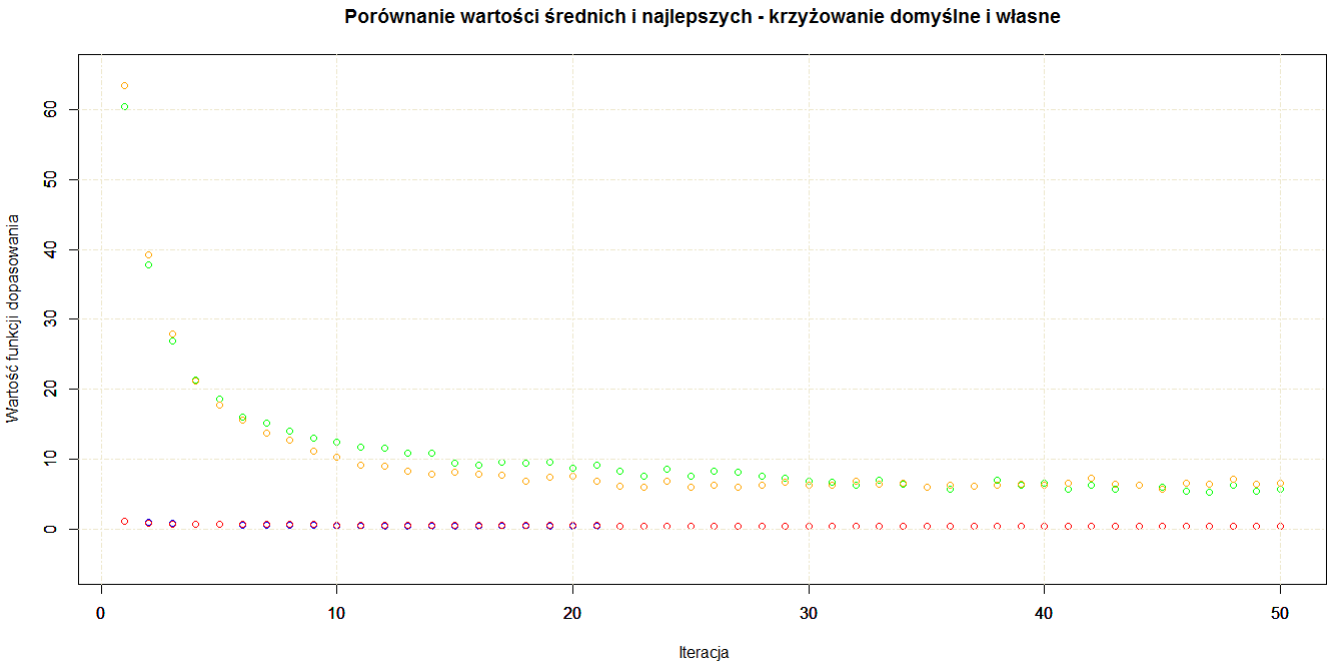
1.6 Krzyżowanie

Krzyżowanie jest podstawowym mechanizmem tworzenia potomstwa na podstawie wartości przyjmowanych przez rodziców. W stworzonej implementacji pobierana jest połowa wartości od każdego z nich, a ich suma stanowi nowego potomka

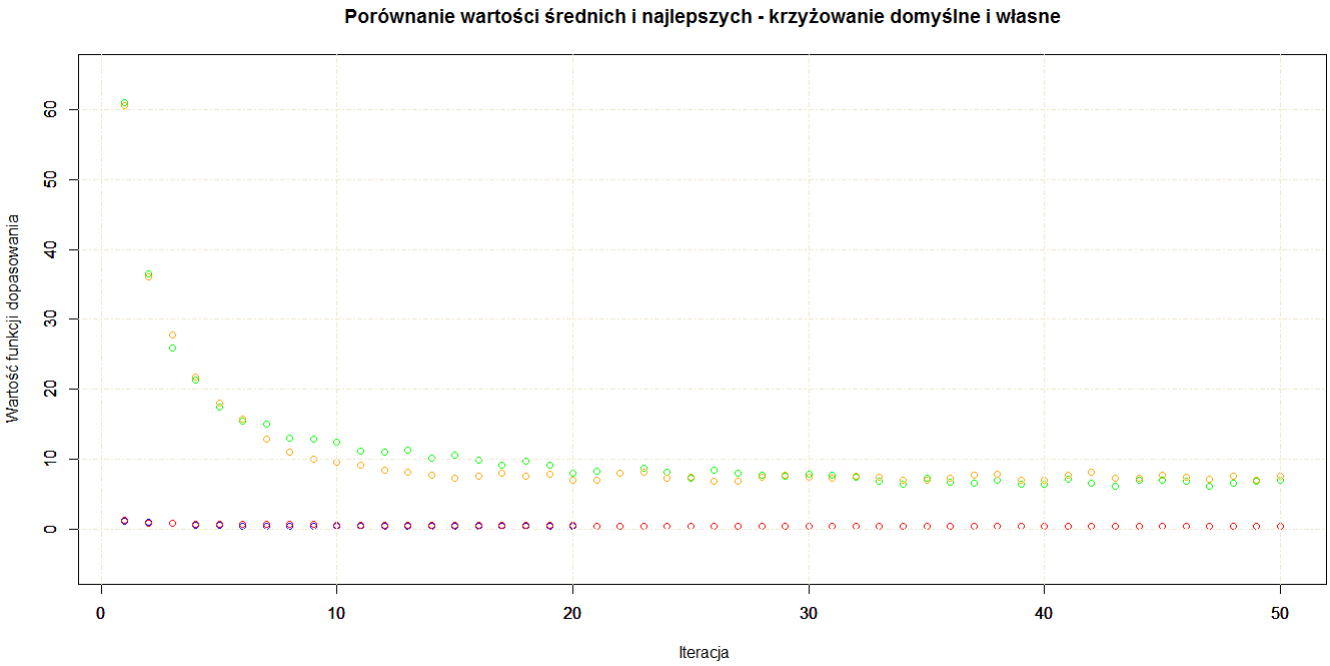
1.6.1 Kod źródłowy

```
myCrossover <- function (ga_object, parents)
{
  # Pobranie rodzicow do skrzyzowania
  # i zainicjalizowanie nimi potomstwa
  parents <- ga_object@population[parents,,drop = FALSE]
  parCol <- ncol(parents)
  parRow <- nrow(parents)
  children <- parents


  for (i in 1:parCol)
  {
        for(j in 1:parRow)
        {
          if (j == parCol) {
            nextCol <- 1
          }
          else {
            nextCol <- j+1
      }

      # Resultat krzyzowania -
      # suma 1/2 wartosci kazdego rodzica z wybranych pol
          newChild <- parents[i, j]/2 + parents[i, nextCol]/2
          children[i, j] <- newChild
        }
  }

  out <- list(children = children, fitness = rep(NA,2))
  return(out)
}
```

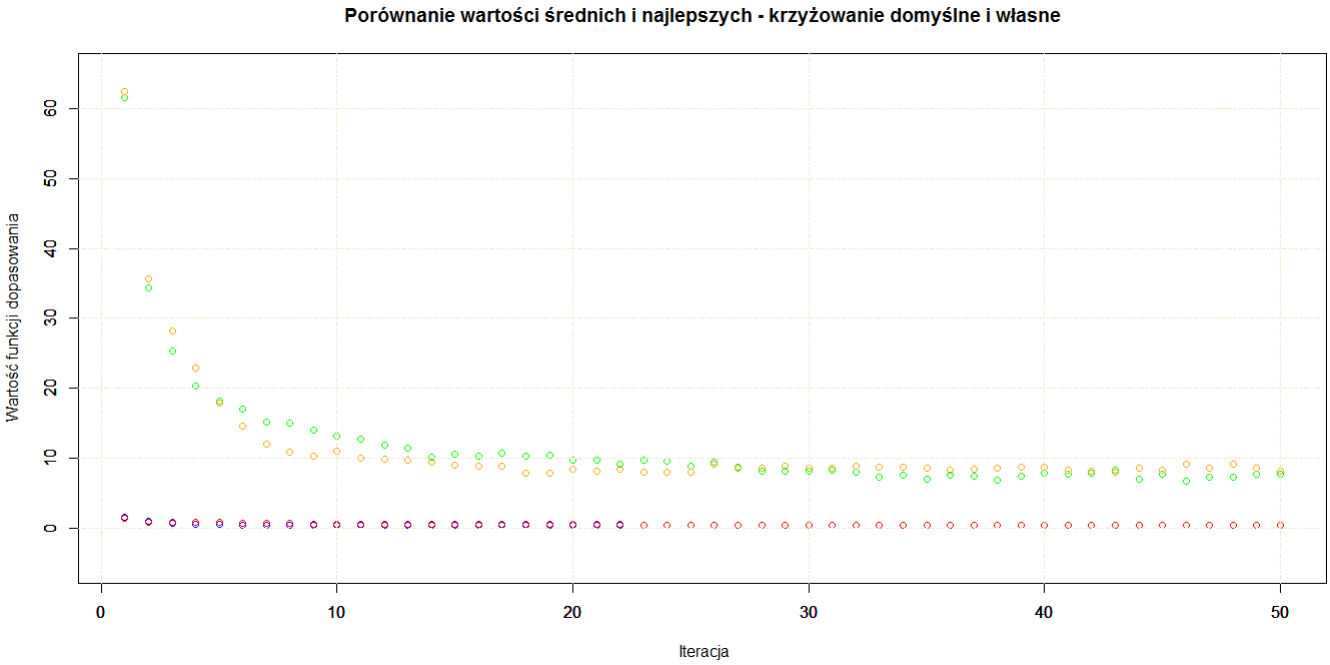
1.6.2 Wyniki badań



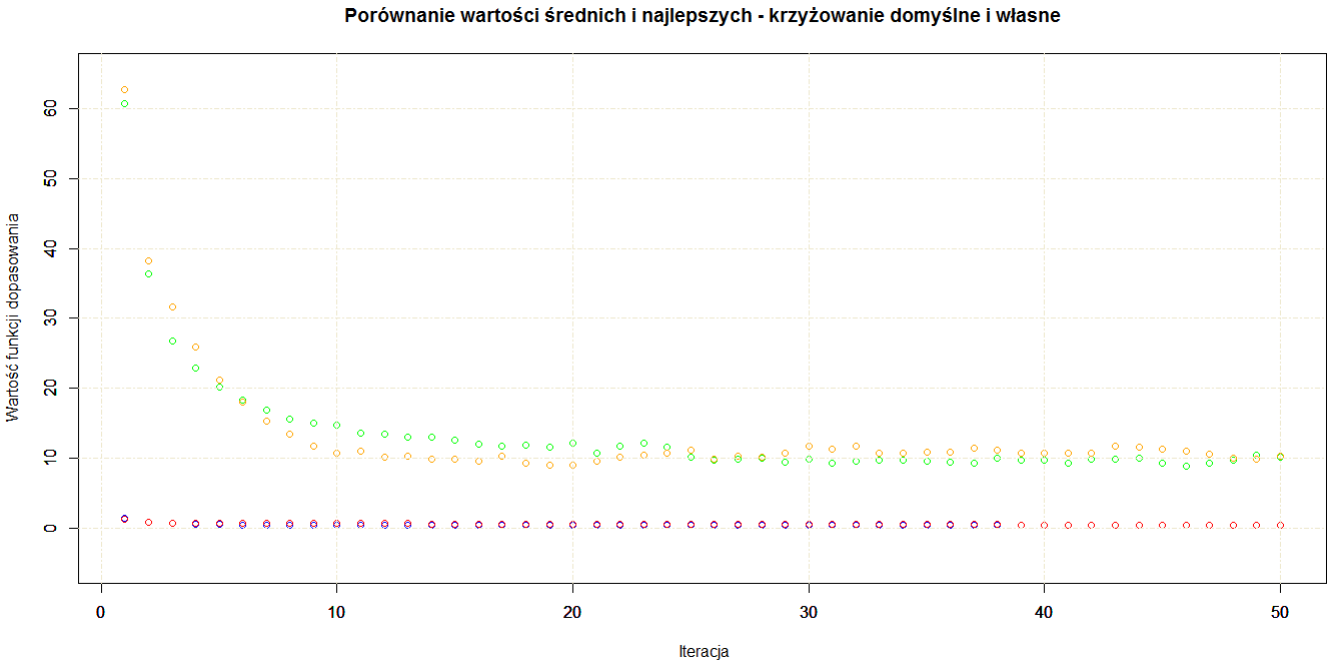
Rysunek 10: Wykres dla prawdopodobieństwa krzyżowania 0.2



Rysunek 11: Wykres dla prawdopodobieństwa krzyżowania 0.5



Rysunek 12: Wykres dla prawdopodobieństwa krzyżowania 0.7



Rysunek 13: Wykres dla prawdopodobieństwa krzyżowania 1

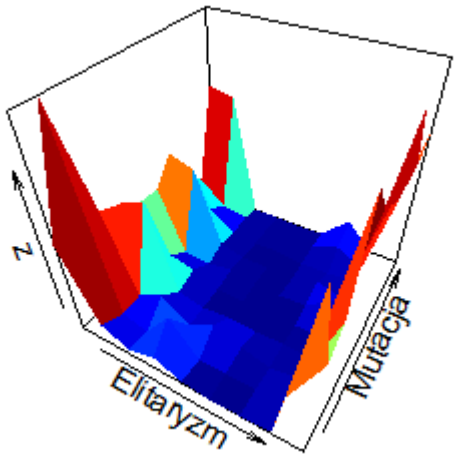
1.6.3 Wnioski

Tabela 3: Wartości średnie i najlepsze osobnika dla domyślnej i własnej funkcji krzyżowania

Prawdopodobieństwo krzyżowania	**Krzyżowanie domyślne**		**Krzyżowanie własne**	
	Wartość średnia	Najlepszy wynik	Wartość średnia	Najlepszy wynik
0.2	5.697605	0.399347	6.538159	0.440933
0.5	6.973119	**0.398064**	7.590295	0.457468
0.7	7.753576	0.398096	8.148286	0.464140
1	10.206810	0.398485	10.375570	0.476180

Tak jak w przypadku mutacji, krzyżowanie nie wpłynęło na poprawę najlepszego wyniku, wręcz przeciwnie - zwiększanie prawdopodobieństwa krzyżowania coraz bardziej pogarszało otrzymywane wartości. Rezultat najbardziej zbliżony do minimum globalnego został uzyskany dla parametrów domyślnych i wbudowanych funkcji.
Średnie populacji dla tych dwóch różnych implementacji przeplatały się, oscylując wokół zbliżonych wartości.

1.7 Badania przy zmianie dwóch parametrów jednocześnie



Rysunek 14: Wykres dla własnej funkcji mutacji i selekcji elitarnej

Na wykresie widać, że skrajne wartości parametrów powodują gorsze wyniki. Należy więc wybierać uśrednione wartości.

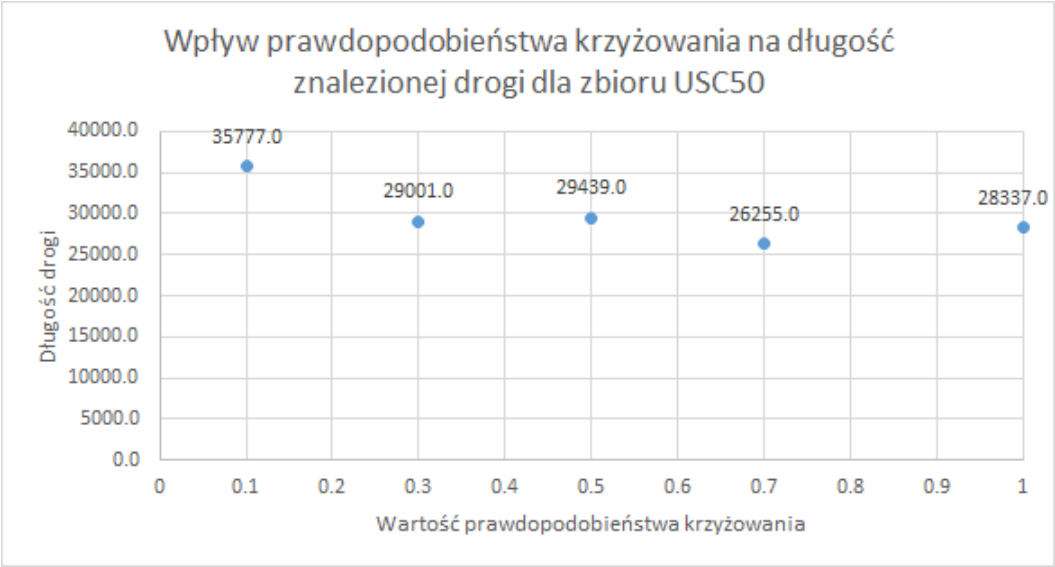
2 Problem komiwojażera

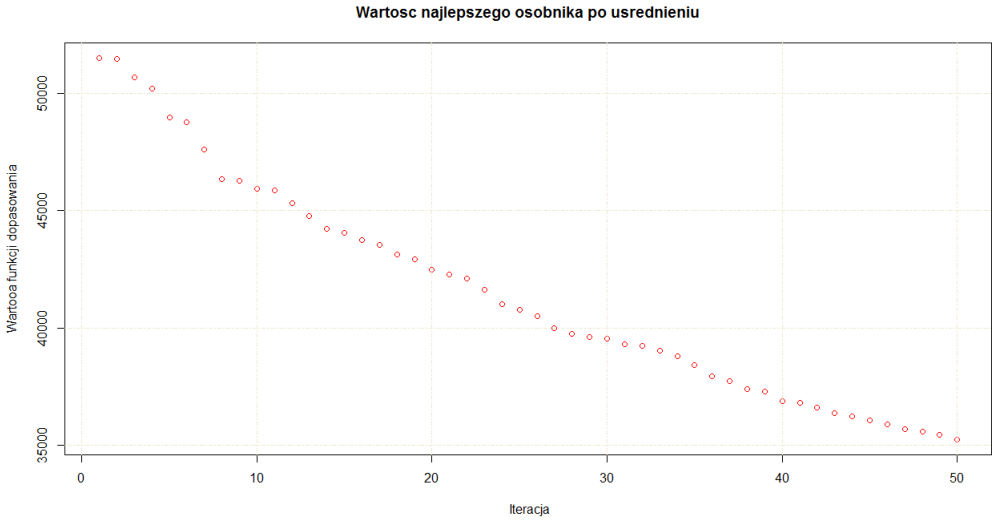
Do badań zostały wykorzystane trzy instancje US50, bay29 i brg180. Wpływ zmiany parametrów potwierdził słuszność domyślnych ustawień pakietu GA. Dodatkowo w tym sprawozdaniu potwierdzają się również wnioski wysnute w sprawozdaniu nr 1.

2.1 Kod źródłowy

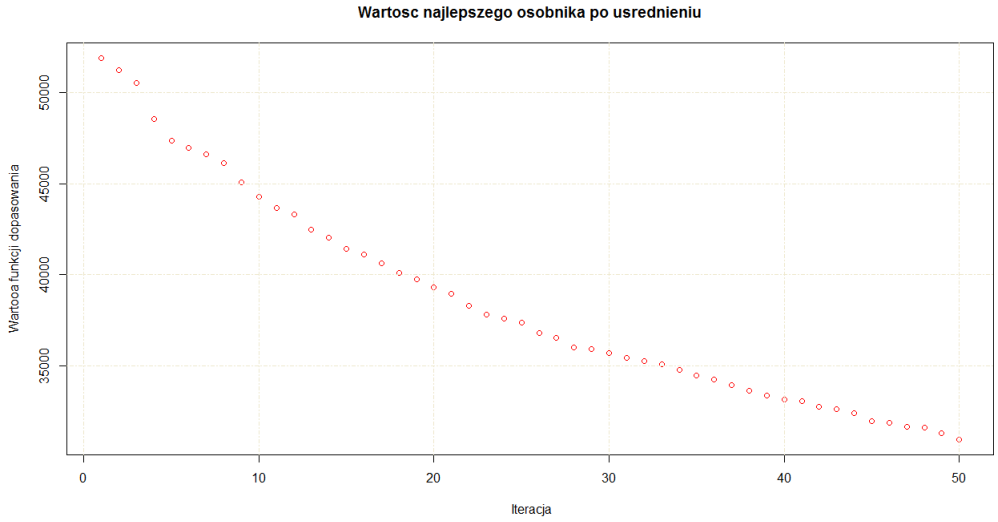
```
data("USCA50")
D <- as.matrix(USCA50)
d <- USCA50


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

tpsFitness <- function(tour, ...) 1/tourLength(tour, ...)

meanGA <- function(population,iteration,crosing,mutant,inst,elit, D)
{
   xMax <- matrix ( 0 ,iteration,15)
   xMean <- matrix ( 0 ,iteration,15)

   for ( i in 1:15)
   {
        GA <- ga(type = "permutation",
        fitness = tpsFitness ,
        min = 1,distMatrix = D, max =attr(inst, "Size"),
        pcrossover = crosing, pmutation = mutant,
        popSize = population, maxiter = iteration,elitism = elit )
        xMax[,i] <- 1/GA@summary[,1]
        xMean[,i] <- 1/GA@summary[,2]
        print(GA@solution)
   }

   return(list(max=xMax,min=xMean))

}
gaResult <- meanGA(100,50,0.5,0.1,d,5,D)
meanRowsMax <- rowMeans(gaResult$max )
meanRowsMean <- rowMeans(gaResult$min)
```

2.2 Krzyżowanie

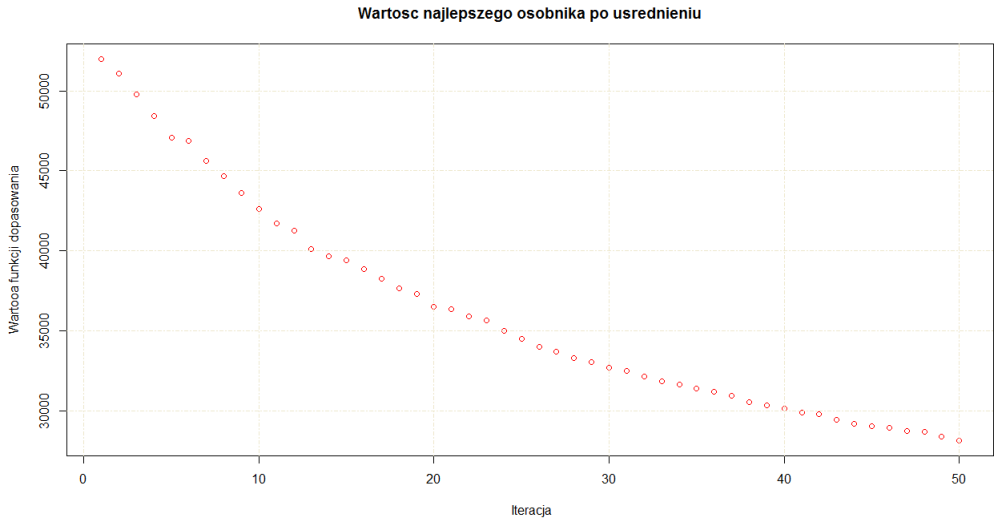




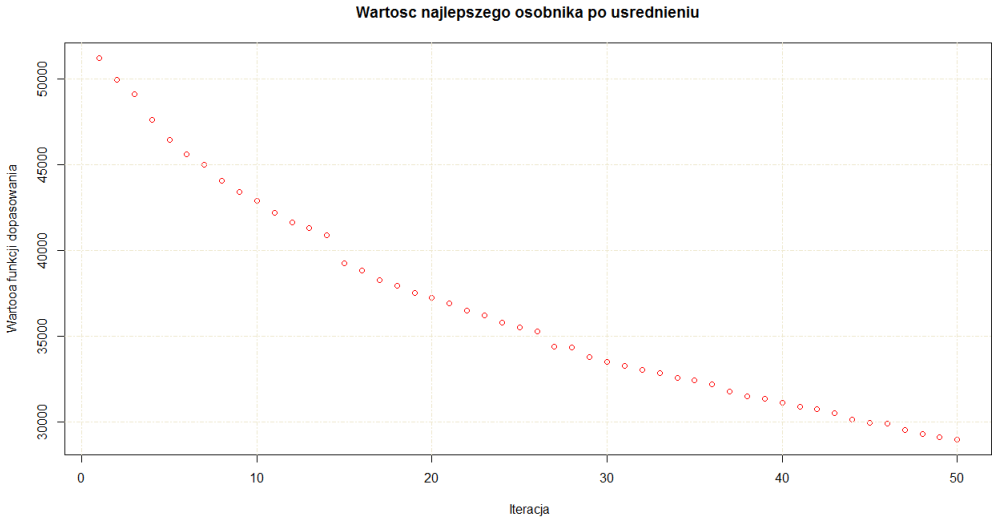
Rysunek 15: Wykres wartości najlepszego osobnika dla prawdopodobieństwa krzyżowania równego 0.1



Rysunek 16: Wykres wartości najlepszego osobnika dla prawdopodobieństwa krzyżowania równego 0.3



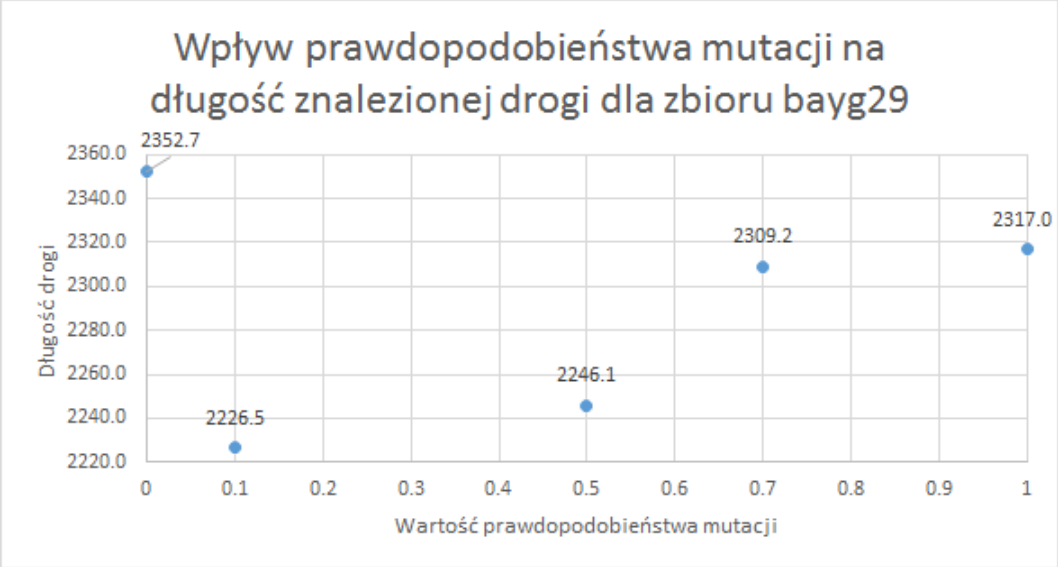
Rysunek 17: Wykres wartości najlepszego osobnika dla prawdopodobieństwa krzyżowania równego 0.7



Rysunek 18: Wykres wartości najlepszego osobnika dla prawdopodobieństwa krzyżowania równego 1.0

Podsumowanie: Najkrótsza droga została znaleziona dla prawdopodobieństwa krzyżowania równego 0.7.

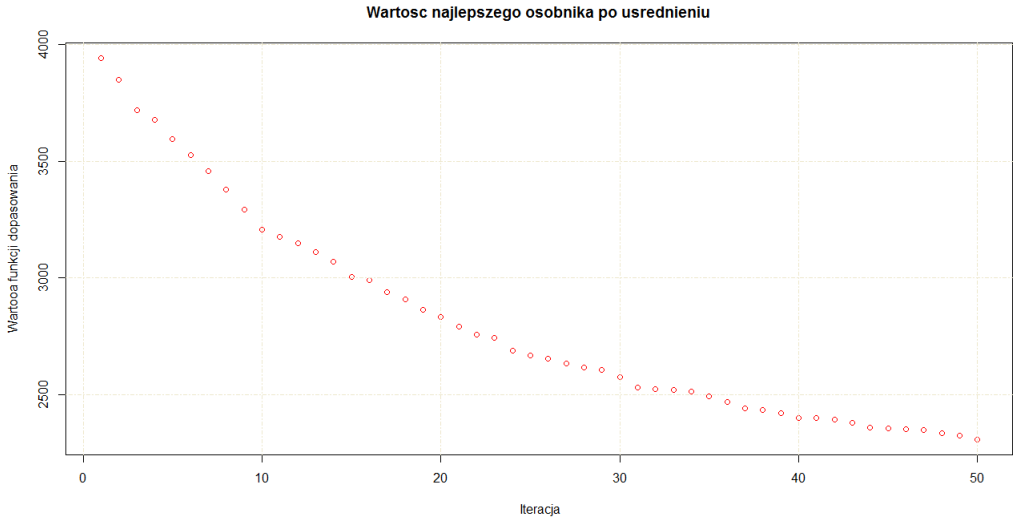
2.3 Mutacja



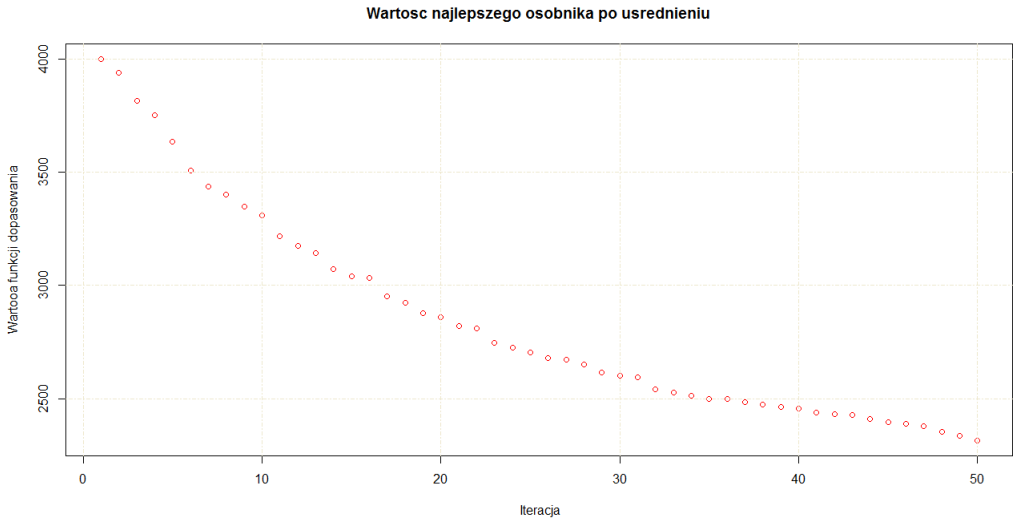
Rysunek 19: Wykres wartości najlepszego osobnika dla prawdopodobieństwa mutacji równego 0

Rysunek 20: Wykres wartości najlepszego osobnika dla prawdopodobieństwa mutacji równego 0.1

Rysunek 21: Wykres wartości najlepszego osobnika dla prawdopodobieństwa mutacji równego 0.5



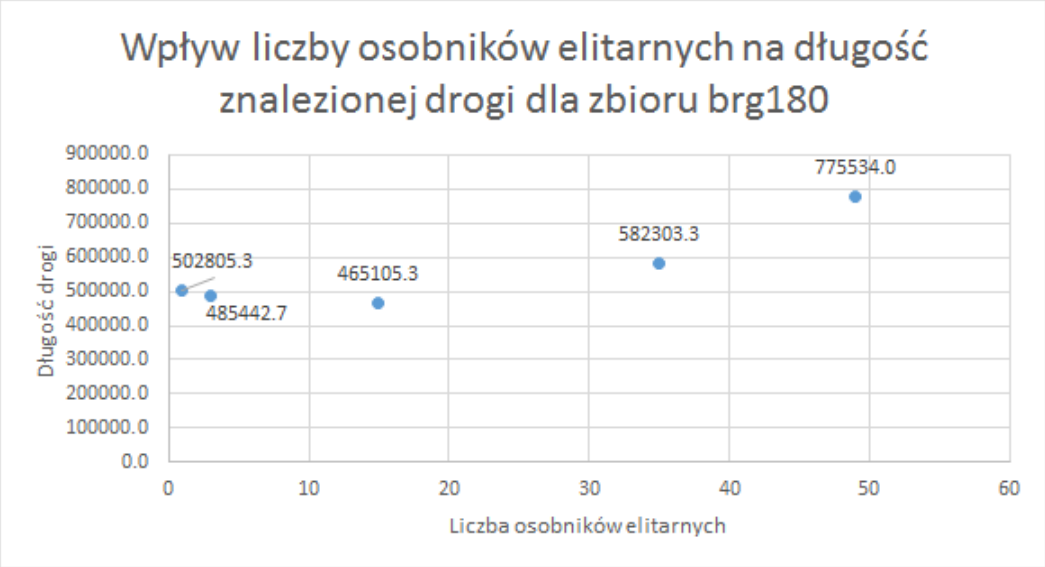
Rysunek 22: Wykres wartości najlepszego osobnika dla prawdopodobieństwa mutacji równego 0.7



Rysunek 23: Wykres wartości najlepszego osobnika dla prawdopodobieństwa krzyżowania równego 1

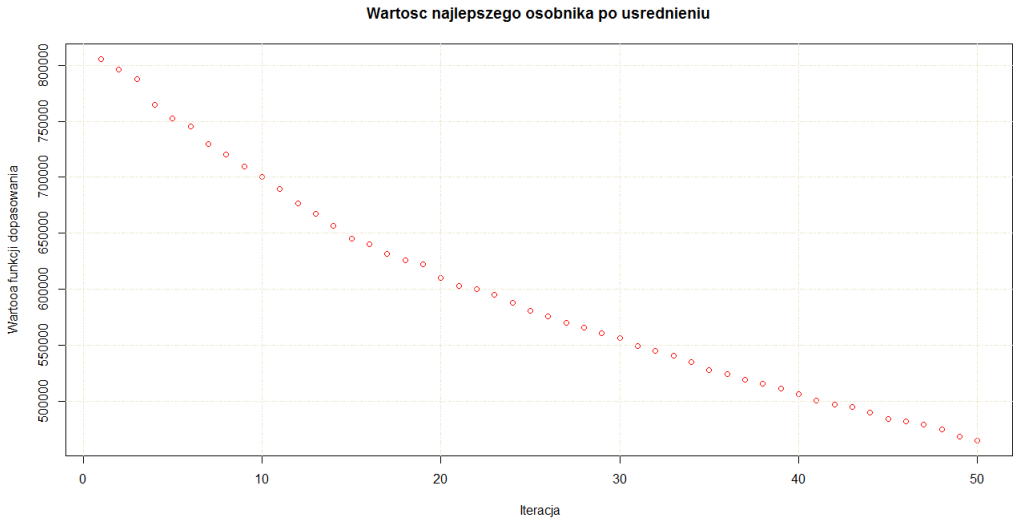
Podsumowanie: Najkrótsza droga została znaleziona dla prawdopodobieństwa mutacji równego 0.1 - jest to jednocześnie wartość domyślna pakietu GA.

2.4 Elitaryzm



Rysunek 24: Wykres wartości najlepszego osobnika dla liczby osobników elitarnych równej 1

Rysunek 25: Wykres wartości najlepszego osobnika dla liczby osobników elitarnych równej 3



Rysunek 26: Wykres wartości najlepszego osobnika dla liczby osobników elitarnych równej 15

Rysunek 27: Wykres wartości najlepszego osobnika dla liczby osobników elitarnych równej 35

Rysunek 28: Wykres wartości najlepszego osobnika dla liczby osobników elitarnych równej 49

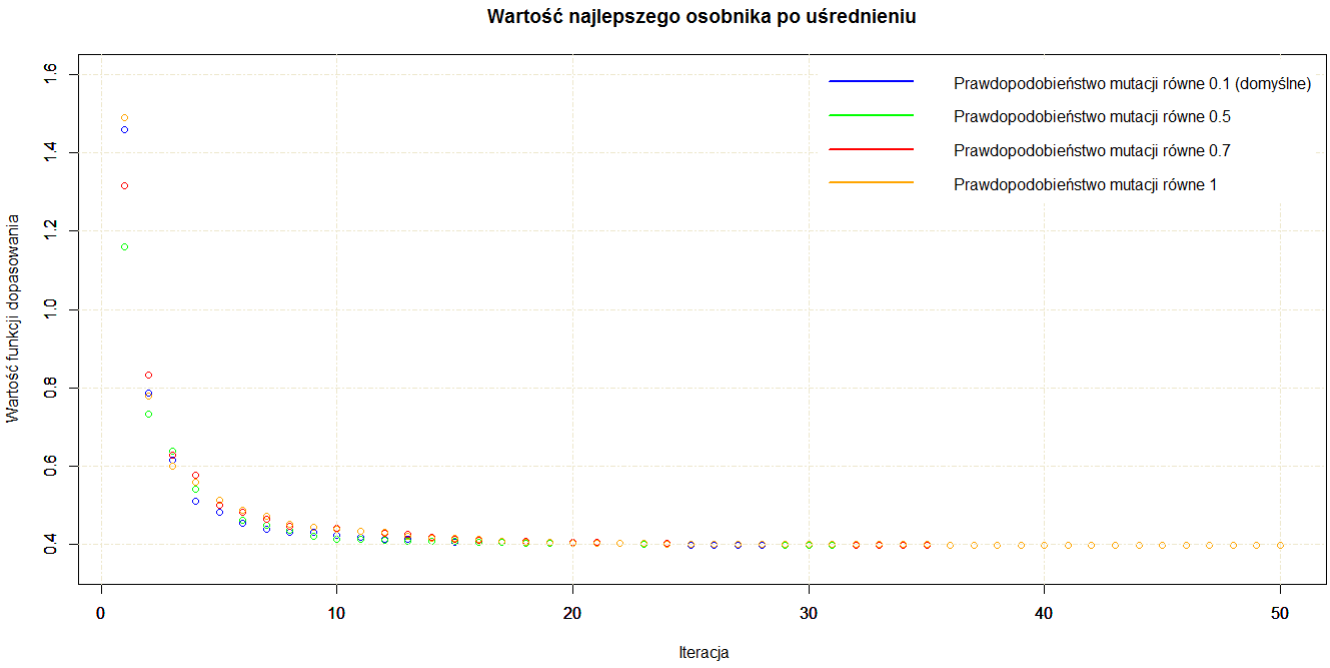
Podsumowanie: Najkrótsza droga została znaleziona, gdy pozostawionych zostało 5 osobników elitarnych z populacji 50.

3 Algorytm genetyczny hybrydowy (memetyczny)

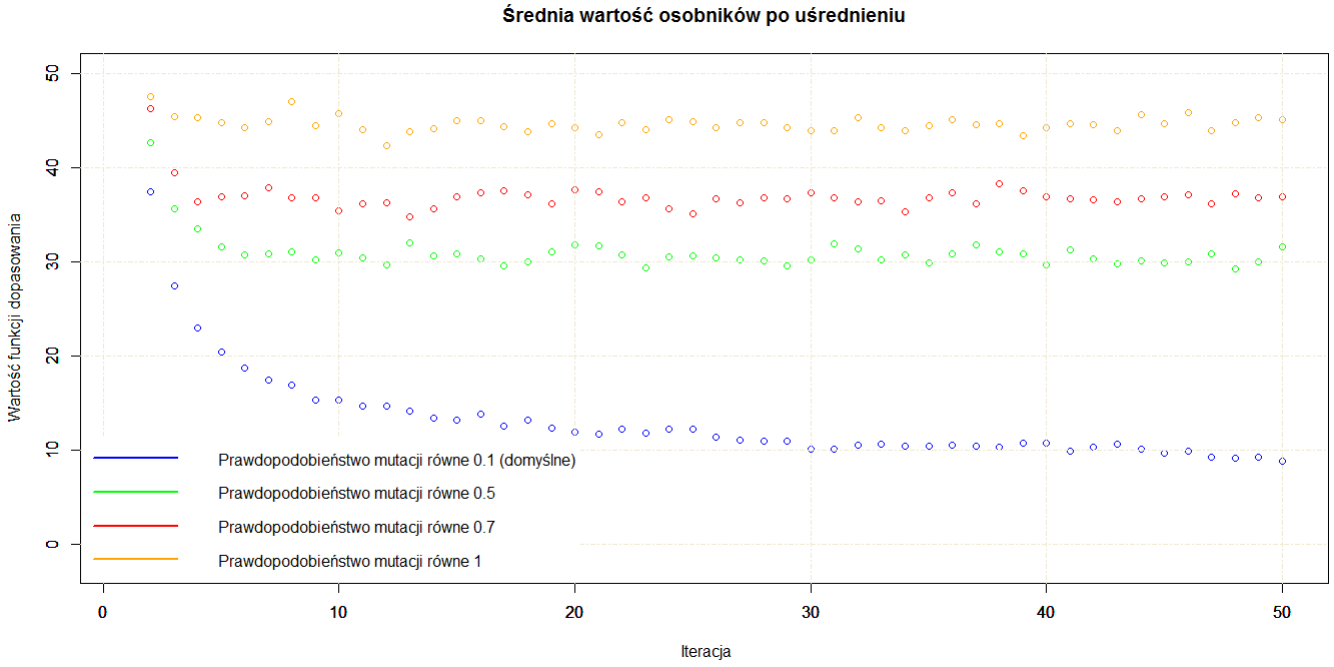
3.1 Wstęp

Algorytmy memetyczne są pewnym ulepszeniem w stosunku do genetycznych. Stosowane są w nich dodatkowo algorytmy wyszukiwania lokalnego, które powinny zwiększać dokładność otrzymanych wyników.
Jeśli chodzi o implementację, uruchomienie algorytmu hybrydowego z wykorzystaniem pakietu *GA* różni się od zwykłego genetycznego jedynie parametrem *optim* ustawionym na *TRUE*.

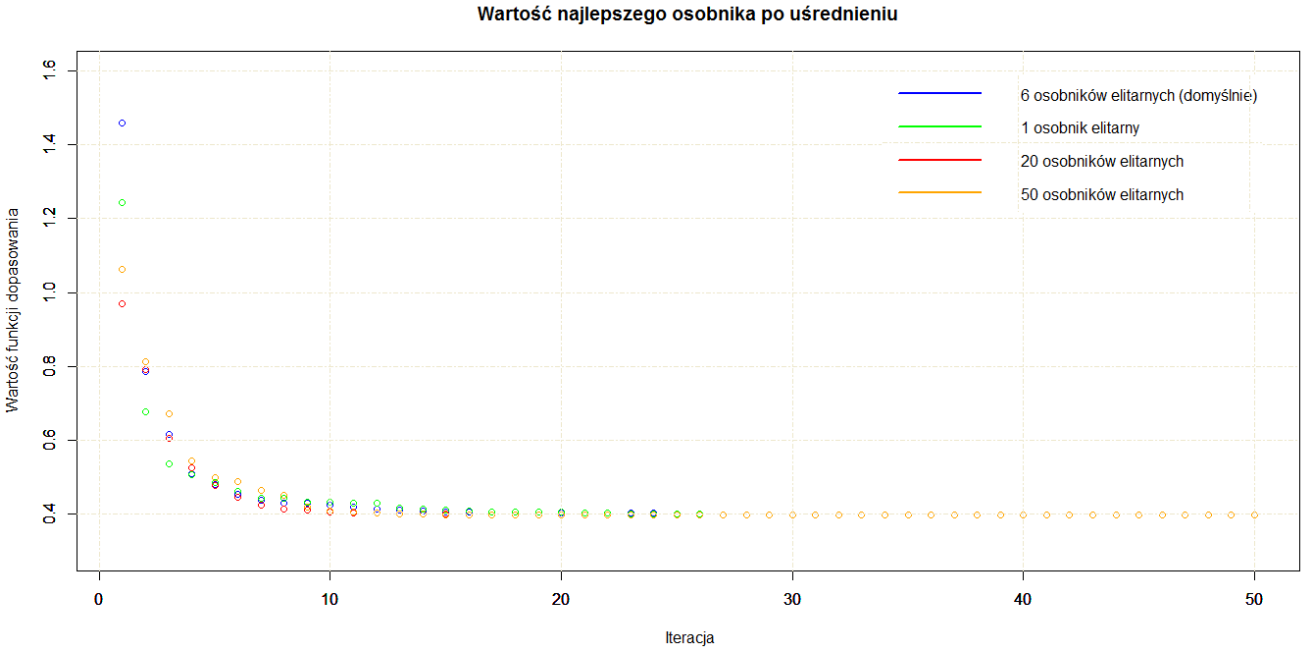
3.1.1 Wyniki badań



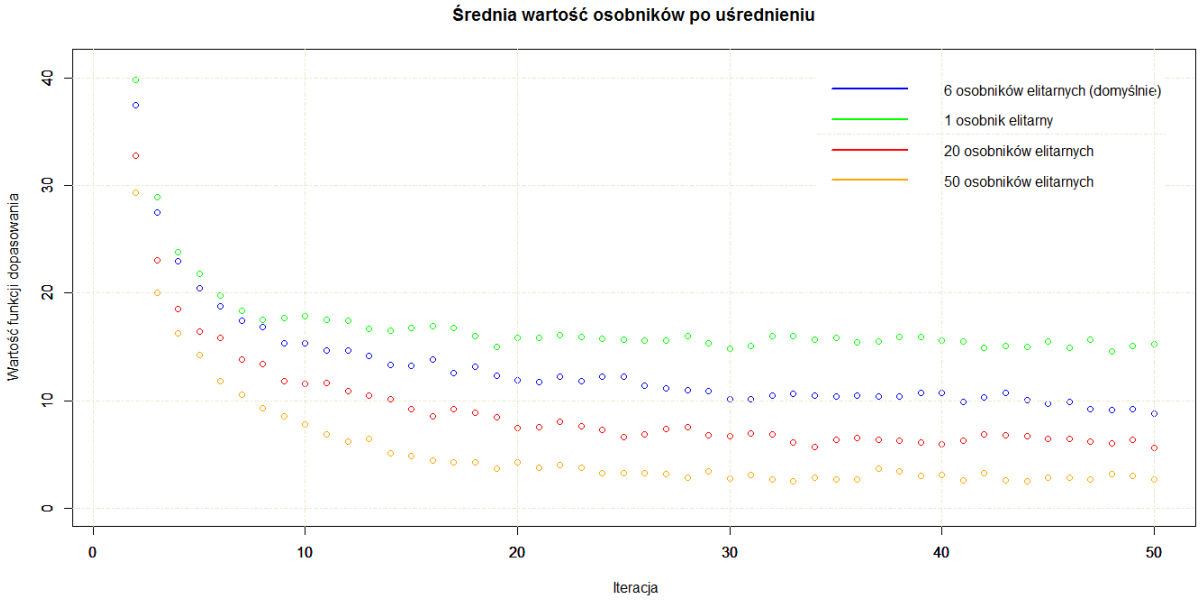
Rysunek 29: Wartości najlepszych osobników dla różnych prawdopodobieństw mutacji



Rysunek 30: Wartości średnie osobników dla różnych prawdopodobieństw mutacji



Rysunek 31: Wartości najlepszych osobników dla różnej ilości osobników elitarnych



Rysunek 32: Wartości średnie osobników dla różnej ilości osobników elitarnych

3.1.2 Wnioski

Tabela 4: Wartości średnie i najlepsze osobnika dla różnych prawdopodobieństw mutacji

Prawdopodobieństwo mutacji	Wartość średnia	Najlepszy wynik
0.1	8.811131	0.398058
0.5	31.624510	**0.397912**
0.7	36.915900	0.397930
1	45.09641	0.398302

Tabela 5: Wartości średnie i najlepsze osobnika dla różnych ilości osobników elitarnych

Liczba osobników elitarnych	Wartość średnia	Najlepszy wynik
6	8.811131	0.398058
1	15.28275	0.398001
20	5.619913	**0.397887**
50	2.657782	**0.397887**

W przypadku selekcji elitarnej udało się uzyskać rezultat identyczny (z dokładnością do 0.000 001) z podanym w dokumentacji minimum lokalnym funkcji *branin*. Tym samym zostały potwierdzone założenia teoretyczne, jakoby algorytm hybrydowy miał polepszyć wyszukiwanie lokalnego extremum.