

POLITECHNIKA WROCŁAWSKA

INTELIGENCJA OBLICZENIOWA I JEJ ZASTOSOWANIA

---

**Badanie algorytmu genetycznego z zakresu  
optymalizacji globalnej dla wybranej  
funkcji testowej. Przeprowadzenie  
pomiarów dla algorytmu hybrydowego i  
optymalizacji rojem cząstek**

---

*Autorzy:*

Paweł ANDZIUL 200648

Marcin SŁOWIŃSKI 200638

*Prowadzący:*

dr hab. inż. Olgierd UNOLD,

prof. nadzw. PWr

26 kwietnia 2017

# Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>2</b>
<b>2</b>	<b>Cel ćwiczeń</b>	<b>3</b>
<b>3</b>	<b>Opis wykorzystanych funkcji</b>	<b>4</b>
<b>4</b>	<b>Opis własnych funkcji</b>	<b>5</b>
<b>5</b>	<b>Wykorzystane narzędzia i biblioteki</b>	<b>6</b>
5.1	RStudio . . . . .	6
5.2	GlobalOptTest . . . . .	6
5.3	DoParallel . . . . .	6
5.4	Rgl . . . . .	6
5.5	GA . . . . .	6
<b>6</b>	<b>Przebieg badań dla problemu optymalizacji rzeczywistej</b>	<b>7</b>
6.1	Gulf . . . . .	7
<b>7</b>	<b>Przebieg badań dla problemu komiwojażera</b>	<b>9</b>
7.1	Algorytm genetyczny . . . . .	9
7.2	Algorytm hybrydowy . . . . .	9
7.3	Optymalizacja rojem cząstek . . . . .	9
<b>8</b>	<b>Podsumowanie</b>	<b>10</b>
<b>9</b>	<b>Implementacja</b>	<b>10</b>

# 1 Wprowadzenie

Algorytm genetyczny – algorytm heurystyczny, który swoim działaniem przypomina działanie ewolucji w naturze. Osobniki będące zbyt słabe zostają wyeliminowane z populacji w kolejnych pokoleniach, a na ich miejsce przyjmowane są lepsze, silniejsze, bardziej podatne adaptacji. Algorytmy te zakładają możliwość mutacji i krzyżowania wśród potomków, przez co nie zawsze są oni silniejsi od poprzednio wyeliminowanych członków. Dodatkowo wprowadzają pojęcie elity, która jest bezpośrednio przenoszona do następnego – teoretycznie lepszego pokolenia.

*dla wybranej funkcji własnej funkcje krzyżowania (dla branina) dla tsp (np-trudny) genetyczny – tsplib wykorzystać do badań (2–3 instancje srednie male duze) z własnym operatorem z domyslnym algorytm ga z lokalnym wyszukiwaniem, dla komiwojażera, założyć czy ma lepsze wartości, czy szybciej zbiega, jak operatory się zachowują, psoptim, dla jednej funkcji i komiwojażera*

W ramach laboratorium należało przeprowadzić testy algorytmu genetycznego dla różnych parametrów. Jako benchmark oceny należało użyć pakietu „getGlobalOpts” oraz języka R.

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

## **2 Cel ćwiczeń**

Celem jest

### 3 Opis wykorzystanych funkcji

Gulf, TSP, hybrydowy

Branin jest funkcją z dwoma parametrami. Na ilustracji (rys. 1) przedstawiono jej wykres a poniżej jej wzór (1) [4].

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

, gdzie  $x_1 \in [-5, 10]$  oraz  $x_2 \in [0, 15]$ .

## 4 Opis własnych funkcji

Opisać jak działa własna funkcja mutacji

## **5 Wykorzystane narzędzia i biblioteki**

### **5.1 RStudio**

### **5.2 GlobalOptTest**

### **5.3 DoParallel**

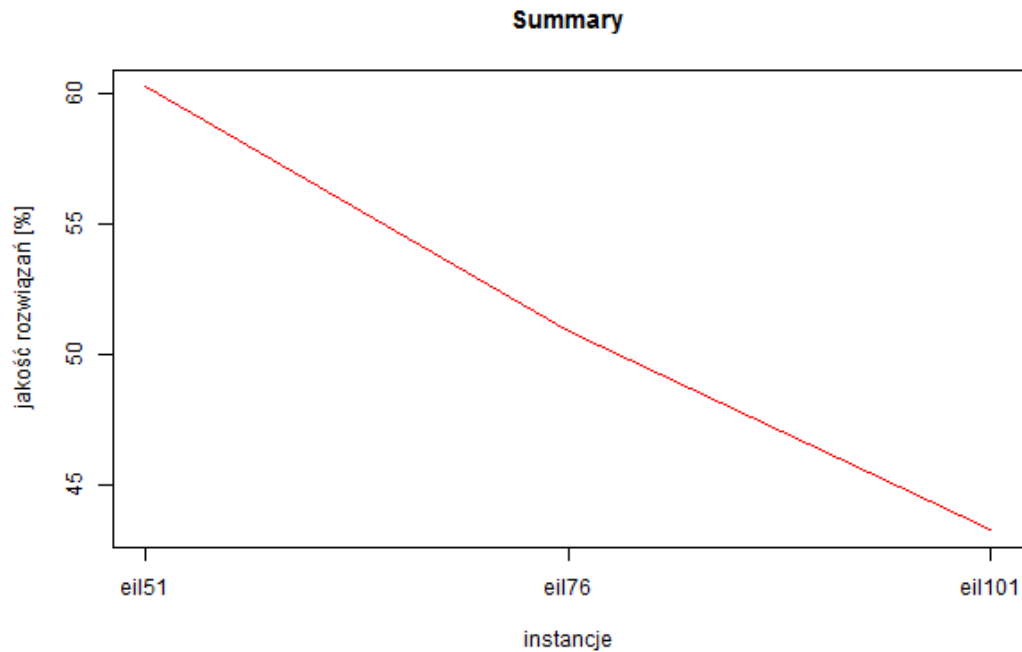
### **5.4 Rgl**

### **5.5 GA**

## 6 Przebieg badań dla problemu optymalizacji rzeczywistej

Do badań wykorzystano następujące funkcje.

### 6.1 Gulf

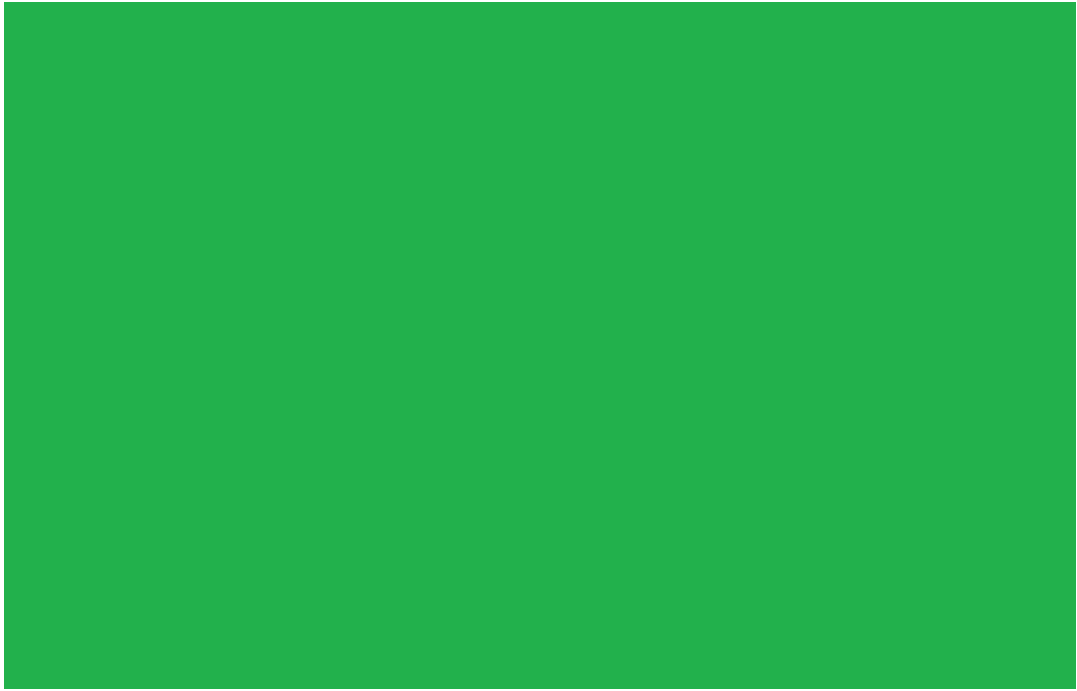


Rysunek 1: Jakość wyników

Z wykresu (rys. 1) wynika, że funkcja ta ma stosunkowo duży obszar w którym może znajdować się minimum oraz dwie strefy w których wartości są dużo większe.

Na kolejnych stronach zamieszczono wyniki pomiarów dla różnych wartości prawdopodobieństwa mutacji dla zmienionej i niezmienionej funkcji mutacji. Pomiary wykonano dla 4 różnych ustawień domyślnych parametrów (serie 1 – 4).





Rysunek 2: Wartość znalezionej minimum funkcji Branin w zależności od prawdopodobieństwa mutacji

## **7 Przebieg badań dla problemu komiwojażera**

Do badań wykorzystano następujące funkcje.

### **7.1 Algorytm genetyczny**

### **7.2 Algorytm hybrydowy**

### **7.3 Optymalizacja rojem cząstek**

## 8 Podsumowanie

W trakcie prowadzonych badań przetestowano algorytm genetyczny w zadaniu optymalizacji dla 9 funkcji testowych. Analizie poddano wpływ zmiany każdego z parametrów dla 4 różnych konfiguracji pozostałych wartości domyślnych.

Wartość prawdopodobieństwa mutacji i krzyżowania zdaje się odgrywać drugorzędną rolę. Istotne jednak by chociaż jedna z nich była włączona z prawdopodobieństwem większym niż 0.

Najlepszym ustawieniem dla elityzmu jest prawdopodobieństwo rzędu 0,5.

Z pewnością należałoby zwiększyć ilość prób poddawanych uśrednianiu gdyż dla przyjętych 20 wyniki ciągle są niestabilne. Warto by również rozważyć pomijanie kilku najlepszych i najgorszych wyników przed uśrednianiem.

Co ciekawe wyniki są widocznie gorsze przy konfiguracji w której krzyżowanie jest wyłączone a p. mutacji wynosi 0,5. Taka prawidłowość objawia się dla wszystkich badanych funkcji.

## 9 Implementacja

Poniżej (listing 1) zamieszczono kod napisany w języku R przygotowany w celu umożliwienia przeprowadzenia pomiarów.

Listing 1: Skrypt w języku R wykorzystany do badań

```
1 # initialize ----
2 # clean old data
3 rm(list=ls())
4 dev.off(dev.list()["RStudioGD"])
5
6 # load libraries
7 require("GA")
8 require("globalOptTests")
9 require("rgl")
10 require("parallel")
11 require("doParallel")
12
13 # custom functions ----
14 # mutation function
15 myMutationFunction <- function(object, parent) {
16   # get GA population
17   population <- parent <- as.vector(object@population[parent, ])
18
19   # calculate randoms
20   rnd <- sample(1:length(population), 1)
21   rndMinOrMax <- sample(1:2, 1)
22
23   # get min and max from population vector
24   max_value <- which.max(population)
25   min_value <- which.min(population)
26
27   # if rndMinOrMax is 0 switch random value to min, else switch to max
28   if (rndMinOrMax == 0)
29   {
30     population[rnd] <- min_value;
```

```

31   } else
32   {
33     population[rnd] <- max_value;
34   }
35
36
37   return (population);
38 }
39 # crossover function
40 myCrossoverFunction <- function(object, parent) {
41
42 }
43
44 # Settings ----
45
46 nOfRuns <- 1 # number of runs to calc avg scores
47
48 numOfCores <- FALSE # number of cores to use (FALSE, 1 - n)
49
50 # colors and titles for plot series
51 colors <- c("red", "blue", "purple", "black")
52 series <- c("Seria 1", "Seria 2", "Seria 3", "Seria 4")
53
54 # default parameters for measurements
55 # each row is a different serie
56 # [mutations,crossovers,populations,iterations,color]
57 params = matrix(
58   c(0, 0, 50, 100, 1,
59     0, 0.8, 50, 100, 2,
60     0.1, 0, 50, 100, 3,
61     0.1, 0.8, 50, 100, 4),
62   nrow=4, ncol=5, byrow = TRUE)
63
64 # names of functions from globalOptTests package
65 functions <- c("Branin", "Gulf")#, "CosMix4", "EMichalewicz",
66   #"Hartman6", "PriceTransistor", "Schwefel", "Zeldasine20")
67
68 # graph settings
69 graphs <- TRUE #true if you want to print graphs
70 quality <- 100 #number of probes
71
72 # sequences of parameters for each serie
73 mutationTests <- seq(0, 1, 0.1)
74 crossoverTests <- seq(0, 1, 0.1)
75 populationTests <- seq(10, 100, 5)
76 iterationTests <- seq(10, 200, 10)
77 elitismTests <- seq(0, 1, 0.1)
78
79 # Processing ----
80
81 customMeasure <- function(fileName, graphName, values, mType, xlab, main) {
82
83   gMin <- .Machine$integer.max
84   gBest <- NA
85
86   # main measurement loop (for each serie and sequence calculate average

```

```

      results)
87 temp <- c()
88 for (defRow in 1:nrow(params)) {
89   averages <- c()
90   for (value in values) {
91     sum <- 0
92     for (i in 1:nOfRuns) {
93       GAmin <- ga(type = "real-valued",
94         mutation = myMutationFunction,
95         #crossover = myCrossoverFunction,
96         fitness = function(xx) -f(xx),
97         min = c(B[1,]), max = c(B[2,]),
98         popSize = if (mType == "pop") value else params[defRow,3],
99         maxiter = if (mType == "itr") value else params[defRow,4],
100        pmutation = if (mType == "mut") value else params[defRow,1],
101        pcrossover = if (mType == "crs") value else params[defRow,2],
102        elitism = if (mType == "elt") value else max(1,
103          round(params[defRow,3] * 0.05)),
104        parallel = numOfCores)
105      solution <- matrix(unlist(GAmin@solution),ncol=dim,byrow=TRUE)
106      eval <- f(solution[1,])
107      if (eval < gMin) {
108        gMin <- eval
109        gBest <- GAmin
110      }
111      sum <- sum + eval
112    }
113    averages <- c(averages, (sum / nOfRuns))
114  }
115  temp <- c(temp, averages)
116 }
117 result <- matrix(c(temp),nrow = nrow(params),ncol = length(values))
118 write.table(result, file = paste(funcName, fileName, sep=""), row.names=FALSE,
119   na="", col.names=FALSE, sep=";")
120
121 if (graphs) {
122   # save graph with measurement series to file
123   png(file = paste(funcName, graphName, ".png", sep=""), width=600,
124     height=400, units="px")
125   plot(0, 0, main=main,
126     ylim=c(min(c(temp,globalOpt)),max(c(temp,globalOpt))),
127     xlim=c(min(values),max(values)),
128     type="n", xlab=xlab, ylab="wartosc")
129   abline(globalOpt,0, col="green")
130   colorNames <- c()
131   seriesNames <- c()
132   for (i in 1:nrow(params)) {
133     color <- colors[params[i,5]]
134     colorNames <- c(colorNames, color)
135     seriesNames <- c(seriesNames, series[params[i,5]])
136     lines(values, result[i,], col = color, type = 'l')
137   }
138   legend("topright", seriesNames, lwd=rep(2,nrow(params)),
139     lty=rep(1,nrow(params)), col=colorNames)
140   dev.off()

```

```

139
140     summary(gBest)
141
142     # save overview of best found minimum to file
143     png(file = paste(funcName, graphName, mType, ".png", sep=""), width=600,
144         height=400, units="px")
145     filled.contour(x, y, z, color.palette = jet.colors, nlevels = 24,
146         plot.axes = { axis(1); axis(2);
147             points(solution[1,1], solution[1,2],
148                 pch = 3, cex = 5, col = "black", lwd = 2)
149         }
150     )
151     dev.off()
152
153     # save best fitness graph to file
154     png(file = paste(funcName, graphName, mType, "fitness", ".png", sep=""),
155         width=600, height=400, units="px")
156     plot(gBest)
157     dev.off()
158 }
159 }
160
161 for (funcName in functions) {
162     # get data from globalOptTests package
163     dim <- getProblemDimen(funcName)
164     B <- matrix(unlist(getDefaultBounds(funcName)), ncol=dim, byrow=TRUE)
165     f <- function(xx) goTest(par=c(xx, rep(0, dim-length(xx))),
166         fnName=funcName, checkDim = TRUE)
167     globalOpt <- getGlobalOpt(funcName)
168
169     if (graphs) {
170         # prepare two versions of graphs (interactive and static)
171         xprobes <- abs(B[2,1] - B[1,1]) / quality
172         yprobes <- abs(B[2,2] - B[1,2]) / quality
173         x <- seq(B[1,1], B[2,1], by = xprobes)
174         y <- seq(B[1,2], B[2,2], by = yprobes)
175         z <- outer(x, y, Vectorize(function(x,y) f(c(x,y))))
176         nbcol = 100
177         color = rev(rainbow(nbcol, start = 0/6, end = 4/6))
178         zcol = cut(z, nbcol)
179         persp3d(x, y, z, theta=50, phi=25, expand=0.75, col=color[zcol],
180             ticktype="detailed", axes=TRUE)
181
182         png(file = paste(funcName, "1.png", sep=""), width=600, height=400,
183             units="px")
184         persp3d(x, y, z, theta = -45, phi = 20, color.palette = jet.colors)
185         dev.off()
186     }
187
188     # for each function perform set of measurements
189     customMeasure("resultsMutations.csv", "2", mutationTests, "mut",
190         "p. mutacji", "Znalezienie minimum dla roznych prawdopodobienstw mutacji")
191     customMeasure("resultsCrossover.csv", "3", crossoverTests, "crs",
192         "p. krzyzowania", "Znalezienie minimum dla roznych prawdopodobienstw
193         krzyzowania")

```

```

191 customMeasure("resultsPopulation.csv", "4", populationTests, "pop",
192               "rozmiar populacji", "Znalezione minimum dla roznych rozmiarow populacji")
193 customMeasure("resultsIterations.csv", "5", iterationTests, "itr",
194               "ilosc iteracji", "Znalezione minimum dla roznych ilosci iteracji")
195 customMeasure("resultsElitism.csv", "6", elitismTests, "elt",
196               "elityzm", "Znalezione minimum dla roznych wartosci elityzmu")
197 }
198
199 # whole source code is located here:
200 # https://github.com/cran/GA/tree/master/R

```

Skrypt przygotowano w sposób który umożliwia w pełni automatyczne przeprowadzenie wszystkich pomiarów. Jednocześnie wszystkie wykresy mogą być natychmiast podmienione w sprawozdaniu. Poniżej pokrótce omówiono podstawowe parametry.

- nOfRuns  
Ilość powtórzeń dla każdego pomiaru w celu uśrednienia.
- colors, series  
Wektory kolorów i nazw kolejnych serii pomiarowych.
- params  
Macierz parametrów domyślnych algorytmu dla każdej z serii. W każdym wierszu kolejno są zawarte: p. mutacji, p. krzyżowania, rozmiar populacji, ilość iteracji oraz kolor serii na wykresach.
- functions  
Wektor nazw funkcji dla których przeprowadzane są kolejno pomiary.

Całość informacji niezbędnych do przeprowadzenia obliczeń odczytywana jest na podstawie nazwy funkcji z pakietu „globalOptTests”. Są to: rozmiar problemu (ilość parametrów), domyślne ograniczenia, wartość w danym punkcie oraz optimum dla domyślnych ograniczeń.

## Literatura

- [1] Artur Suchwałko „Wprowadzenie do R dla programistów innych języków”  
<https://cran.r-project.org/doc/contrib/R-dla-programistow-innych-jezykow.pdf>
- [2] Luca Scrucca „Package GA” <https://cran.r-project.org/web/packages/GA/GA.pdf>
- [3] Surjanovic, S. & Bingham, D. (2013). „Virtual Library of Simulation Experiments: Test Functions and Datasets.” Retrieved April 3, 2017, from <http://www.sfu.ca/ssurjano>.
- [4] Momin Jamil, Xin-She Yang „A literature survey of benchmark functions for global optimization problems”, Int. Journal of Mathematical Modelling and Numerical Optimisation, Vol. 4, No. 2, pp. 150–194. (2013)
- [5] Ajith Abraham, Aboul-Ella Hassanien, Patrick Siarry, Andries Engelbrecht, „Foundations of Computational Intelligence Volume 3” (2009)
- [6] Onay Urfalioglu, Orhan Arikan „Self-adaptive randomized and rank-based differential evolution for multimodal problems” (2011)