Intelligent Systems Seminar Assignment 1

Aljaž Potočnik

15.11.2021

```r
library(GA)
number_and_operators = c("10", "25", "100", "5", "3", "+", "-", "/", "*")
target = 2512
n = length(number_and_operators)
##to get the last index of numbers in vector
nums_end_index = sum(!is.na(as.integer(stringr::str_extract(number_and_operators, "\\d+"))))
```

Population

For creating initial population I used permutation as vector presentation of individual. My implementation makes sure that every number is used only once and operators are choosen randomly.

```r
myInitPopulation <- function(object)
{
  population <- matrix(NA, nrow = object@popSize, ncol = n)
  for(j in 1:object@popSize){
    numbers = 1:nums_end_index
    operators = (nums_end_index+1):n
    p=replicate(n,0)
    for(i in 1:n){
      if(i%%2 == 1){
        temp <- sample(numbers,1)
        if(length(numbers) == 1)p[i]=numbers
        else p[i] <- temp
        numbers <- numbers[-which(numbers==temp)]

      }else{
        temp <- sample(operators,1)
        p[i] <- temp
      }
    }
    population[j,] = p
  }
  population
}
```

Fitness function

I choose a simple fitness function, that is minimizing the difference between target value and individuals.

```r
fitness <-function(object)
{
  str = paste(number_and_operators[object], collapse="")
  x = eval(parse(text=str))
  -abs(target-x)

}
```

Crossover function

My implementation of crossover chooses one point for crossover, takes first part of parents for each child, and takes remaining from other parent in that way that we still get valid equation.If value in vector is not defined we use random sample so we get valid equation.

```
crossover <- function(object, parents)
{
  parents <- object@population[parents,,drop = FALSE]
  n <- ncol(parents)
  #
  cxPoints <- sample(seq(2,n-1), size = 1)
  cxPoints <- seq(0, cxPoints)
  children <- matrix(as.double(NA), nrow = 2, ncol = n)
  children[,cxPoints] <- parents[,cxPoints]
  #
  for(j in 1:2)
  {
    diff = setdiff(parents[-j,], children[j, cxPoints])
    nums = diff[diff <= nums_end_index]
    nums_random_options <- setdiff(setdiff(1:nums_end_index, children[j, children[j] <= nums_end_index]]
    ops = diff[diff > nums_end_index]
    for(i in max(cxPoints+1):n){
      if(i%%2 == 1){
        if(length(nums) == 0)
        {
          children[j, i] =  nums_random_options[1]
          nums_random_options <- nums_random_options[-1]
        }else{
          children[j, i] = nums[1]
          nums <- nums[-1]
        }
      }
      else {
        if(length(ops) == 0)
        {
          children[j, i] =  sample((nums_end_index+1):n, 1)
        }else{
          children[j, i] = ops[1]
          ops <- ops[-1]
        }
      }
    }
  }
  #
  out <- list(children = children, fitness = rep(NA,2))
  return(out)
}
```

Mutation function

My mutation function is pretty similar to the one from library, only it switches two elements which are both numbers of both operators so we get a valid equation.

```
mutation <- function(object, parent)
{
  mutate <- parent <- as.vector(object@population[parent,])
  n <- length(parent)
  repeat{
    m <- sample(1:n, size = 2)
    if(m[1]%%2 == m[2]%%2)
      break
  }
  mutate[m[1]] <- parent[m[2]]
  mutate[m[2]] <- parent[m[1]]
  return(mutate)
}
```

Evaluation

My random search function runs 5 times and takes average of time needed for finding solution:

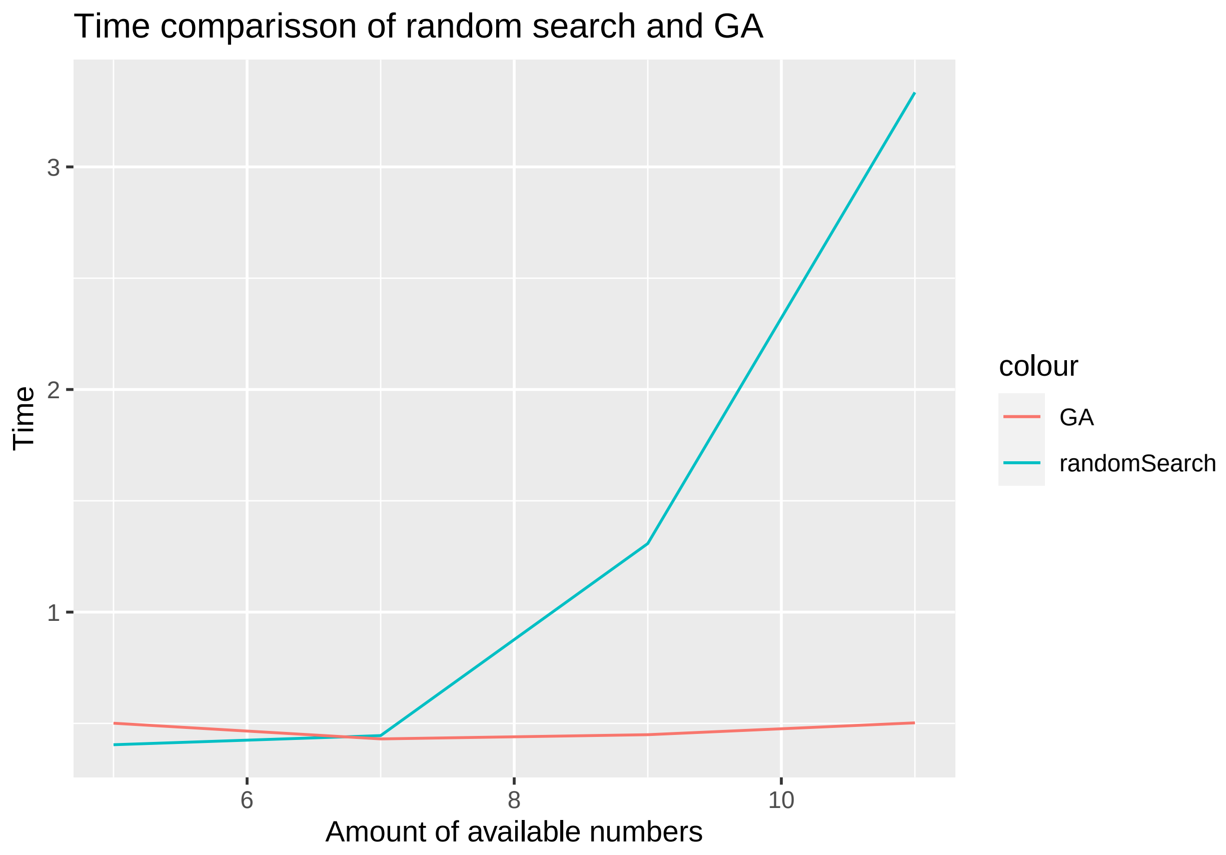
```
randomSearch <- function() {
  time_random <- 0
  for(j in 1:5){
    start_time_ran <- Sys.time()
    end_time_ran <- Sys.time()
    repeat {
      x=replicate(n,0)
      nums = 1:nums_end_index
      for(i in 1:n){
        if(i%%2==1){
          x[i] <- sample(nums,1)
          nums[-which(nums==x[i])]
        }
        else{
          x[i] <- sample((nums_end_index+1):n,1)
        }
      }
      str = paste(number_and_operators[x], collapse="")
      x = eval(parse(text=str))
      if(abs(target-x) == 0){
        end_time_ran <- Sys.time()
        time_random = time_random + (end_time_ran - start_time_ran)
        break
      }
    }
  }
  return(time_random/10)
}
```

Time comparison of random search and GA. Multiple test cases with varying amounts of available numbers :

```r
library(ggplot2)

steviloMeritev <- 4
GA_times <- vector()
random_times <- vector()


for(i in 1:steviloMeritev){
  if(i == 1)number_and_operators=c("10", "25", "100", "5", "3", "+", "-", "/", "*")
  if(i == 2)number_and_operators=c("10", "25", "100", "5", "3","6", "2", "+", "-", "/", "*")
  if(i == 3)number_and_operators=c("10", "25", "100", "5", "3","6", "7","4","11",  "+", "-", "/", "*")
  if(i == 4)number_and_operators=c("10", "25", "100", "5", "3","6", "7", "8","2","4","11", "+", "-", "/
  target = 2512
  n = length(number_and_operators)
  nums_end_index = sum(!is.na(as.integer(stringr::str_extract(number_and_operators, "\\d+"))))
  start_time_GA <- Sys.time()
  GA <- ga(type = "permutation", fitness = fitness, lower = replicate(n,1),
           upper = replicate(n,n),
           popSize = 100, maxiter = 100, run = 50, population = myInitPopulation,
           crossover = crossover, mutation = mutation, selection = ga_tourSelection)
  end_time <- Sys.time()
  GA_time <- end_time - start_time_GA
  GA_times <- c(GA_times, GA_time)
  random_times <- c(random_times, randomSearch())
}
x=seq(5,11,2)
df <- data.frame(x, random_times, GA_times)
ggplot(df, aes(x))+
  ggtitle("Time comparisson of random search and GA")+
  labs(y="Time", x="Amount of available numbers")+
  geom_line(aes(y=random_times, color="randomSearch"))+
  geom_line(aes(y=GA_times, color="GA"))
```



Here we have two lines, one is red that is the line of GA's and blue one is line of random search at different amounts of numbers. We see that more numbers there are longer it takes random search to find solution, and GA stays pretty much the same. Even though in small amounts of numbers random performs better, which is not surprisingly.

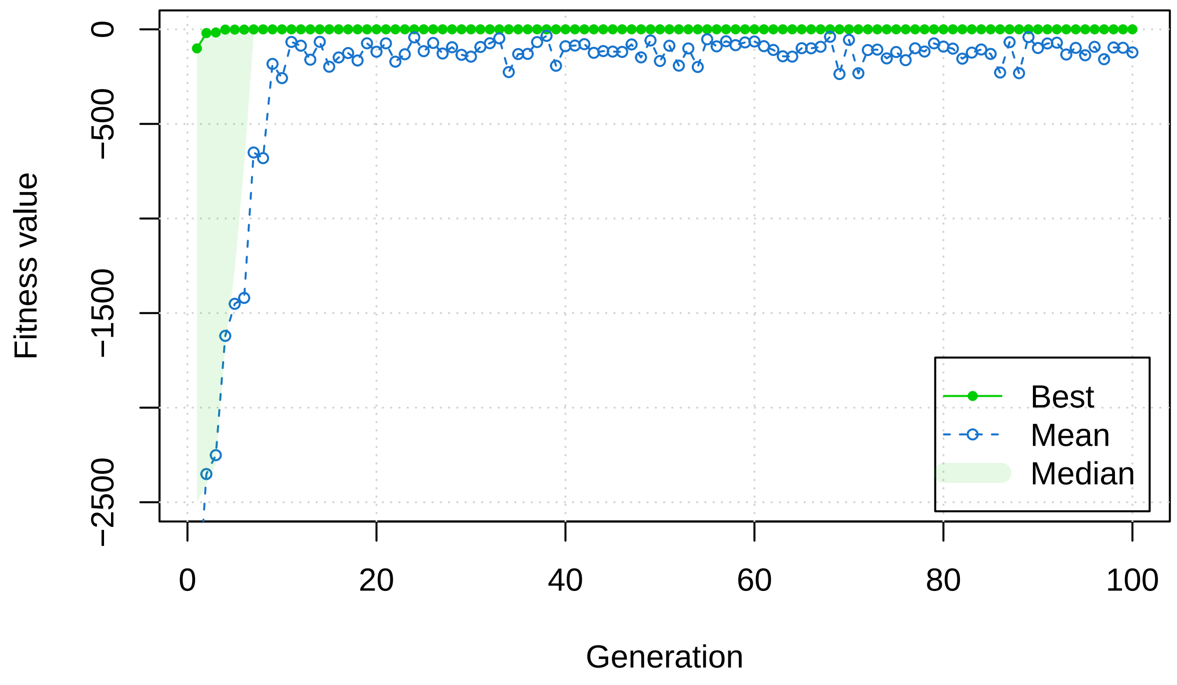
Different configurations

```
GA1 <- ga(type = "permutation", fitness = fitness, lower = replicate(n,1),
         upper = replicate(n,n),
         popSize = 200, maxiter = 100, run = 100, population = myInitPopulation,
         crossover = crossover, mutation = mutation, selection = ga_tourSelection)

summary(GA1)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  200
## Number of generations =  100
## Elitism               =  10
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = 0
## Solutions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x14 x15
##  [1,]  2 15  3 12 11 13  6 14  5  14        12   9
##  [2,]  2 15  3 12 11 13  6 14  9  12        14   1
##  [3,]  3 15  2 12 11 13  6 14  5  14        12   9
##  [4,]  2 15  3 12 11 13  6 14  9  13        15  10
##  [5,]  2 15  3 12 11 13 10 14  8  15        12   9
##  [6,]  3 15  2 12 11 13  6 14  9  12        14   1
##  [7,]  2 15  3 12 11 13  6 14  8  15        12   9
##  [8,]  3 15  2 12 11 13  6 14  8  14        12   9
##  [9,]  2 15  3 12 11 13  6 14  9  13        15   5
## [10,]  3 15  2 12 11 13 10 14  8  14        12   9
## [11,]  3 15  2 12 11 13  6 14  8  15        12   9
```

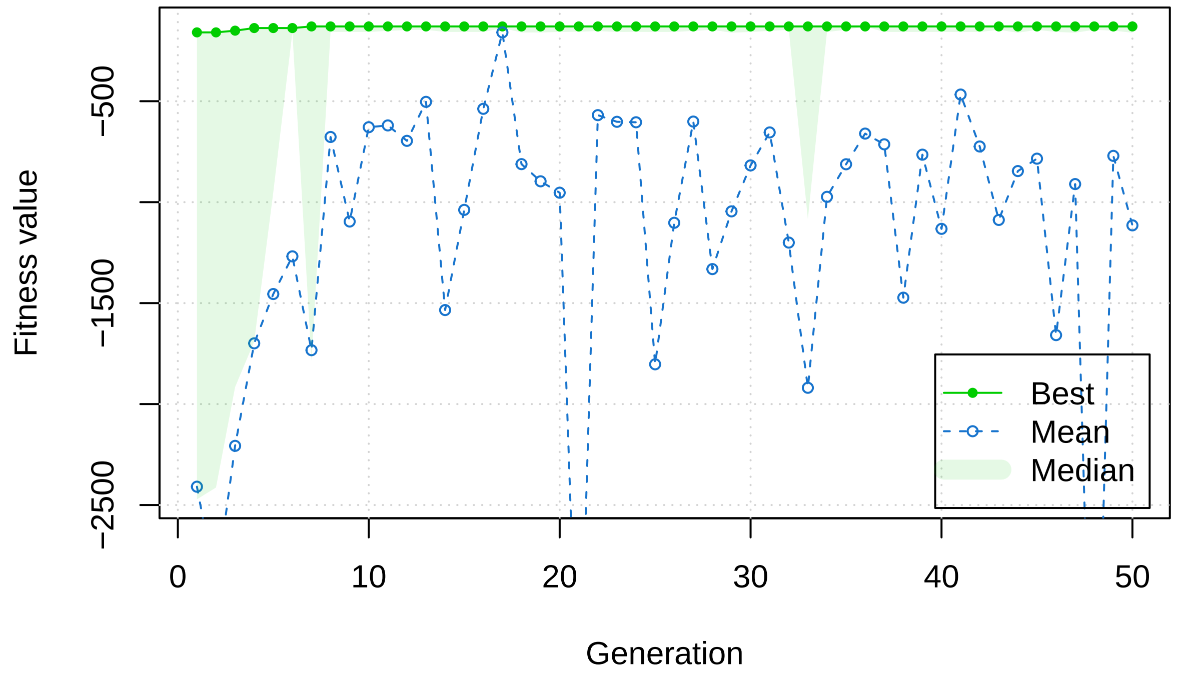
```
plot(GA1)
```



```
GA2 <- ga(type = "permutation", fitness = fitness, lower = replicate(n,1),
         upper = replicate(n,n),
         popSize = 30, maxiter = 50, run = 50, population = myInitPopulation,
         crossover = crossover, mutation = mutation, selection = ga_tourSelection)
summary(GA2)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  30
## Number of generations =  50
## Elitism                =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 50
## Fitness function value = -129.6667
## Solutions =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x14 x15
## [1,]  1 15  9 15  8 15  3 14  6  12        15   7
## [2,]  3 15  9 15  1 15  8 14  6  12        15   7
```

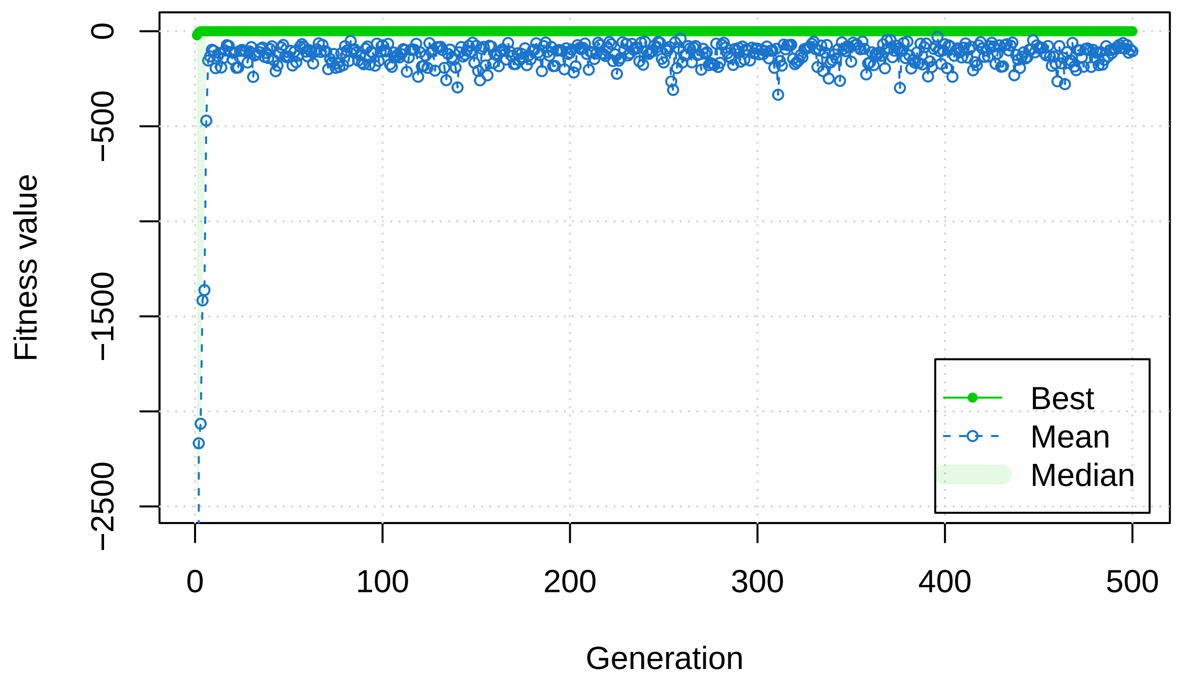
```
plot(GA2)
```



```
GA3 <- ga(type = "permutation", fitness = fitness, lower = replicate(n,1),
         upper = replicate(n,n),
         popSize = 300, maxiter = 500, run = 500, population = myInitPopulation,
         crossover = crossover, mutation = mutation, selection = ga_tourSelection)
summary(GA3)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                   =  permutation
## Population size        =  300
## Number of generations =  500
## Elitism                =  15
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 500
## Fitness function value = 0
## Solutions =
##        x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x14 x15
## [1,]    3 15  2 12 11 12  8 13  1  14        15   7
## [2,]    2 15  3 12  9 12  8 12 11  13        13   7
## [3,]    3 15  2 12 11 12  8 15  1  14        13   7
## [4,]    2 15  3 12  8 12 11 13  9  14        15   7
## [5,]    2 15  3 12  8 12 11 13  1  14        14   4
## [6,]    2 15  3 12 11 12  8 13  1  14        13   7
## [7,]    3 15  2 12 11 13  8 14  1  14        14   4
## [8,]    2 15  3 12 11 12  9 13  1  14        13   7
## [9,]    2 15  3 12  8 12 11 13  1  14        13   7
## [10,]   3 15  2 12 11 13  8 14  1  15        12   7
##  ...
## [22,]   3 15  2 12 11 12  8 12  9  13        13   7
## [23,]   3 15  2 12 11 12  8 13  1  14        13   7
```

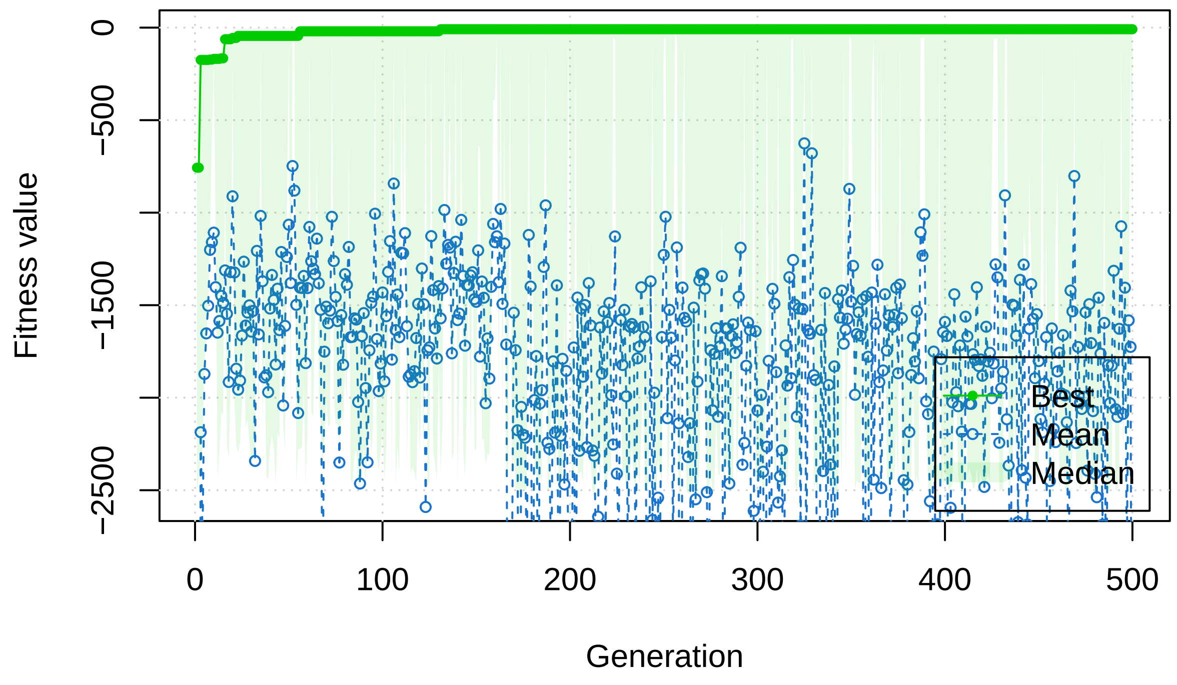
```r
plot(GA3)
```



```r
GA4 <- ga(type = "permutation", fitness = fitness, lower = replicate(n,1),
         upper = replicate(n,n),
         popSize = 30, maxiter = 500, run = 500, population = myInitPopulation,
         crossover = crossover, mutation = mutation, selection = ga_tourSelection)
summary(GA4)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  30
## Number of generations =  500
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 500
## Fitness function value = -8.666667
## Solutions =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x14 x15
## [1,] 10 12  9 15 11 14  6 15  7  15        15   4
## [2,] 10 12 11 15  9 14  6 15  7  15        15   4
```

```
plot(GA4)
```



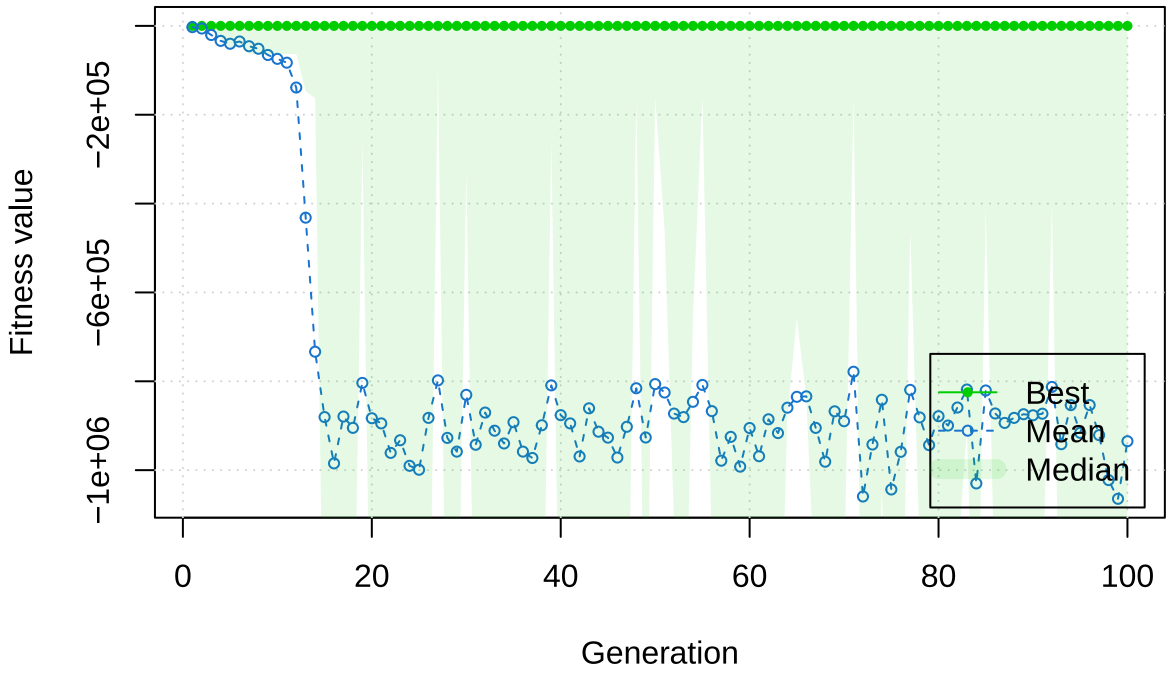
```
GA5 <- ga(type = "permutation", fitness = fitness, lower = replicate(n,1),
         upper = replicate(n,n),
         popSize = 200, maxiter = 100, run = 100, population = myInitPopulation,
```

```r
         crossover = crossover, mutation = mutation, selection = ga_rwSelection)

summary(GA5)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  200
## Number of generations =  100
## Elitism               =  10
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -0.2727273
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x14 x15
## [1,]  3 15  2 13  6 12  1 14 11  15        12  10
```

```r
plot(GA5)
```



I plotted and summed above algorithms. Here we see four different configurations of GA function, they are varying in population size, max iteration, and run parameter. The biggest factor is the size of population. We can see that in GA with smallest population (GA2), we do not reach optimal solution. And also we can see that in GA4 we also do not reach optimal solution even though we have lots of generations. In my opinion algorithms also prematurely converge, we come to solution very soon.In example of GA3 it takes lots more to compute than others event though multiple test did not provide any better accuracy than GA1. Which I think is in my case the best set of parameters.All above GA's selection method is tournament selection except GA5 whose selection method is roulette wheel, and it gives us pretty strange graph, where the mean is more and more away from target.