Assignment1

Libraries

```
library(GA)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.2.2
## Type 'citation("GA")' for citing this R package in publications.

##
## Attaching package: 'GA'

## The following object is masked from 'package:utils':
##
##     de
```

Population

```
numbers_and_operators <- c("10", "25", "100", "5", "3", "+", "-", "/", "*")
operator_min <- 6
operator_max <- 9
n_numbers <- 5
expression_length <- n_numbers * 2 - 1
target_value <- 2512
```

Generate population

This function generates a random agent represented by indices of `numbers_and_operators`.

Agent is generated by choosing 4 random operators (+. -, /, *) with possible repetition and by random permutation of numbers (10.25,100,5, 3).

```
generate_agent <- function(){
  operator_indices <- round(runif(n_numbers - 1, operator_min, operator_max))
  number_indices <- sample(1:n_numbers, n_numbers)
  indices <- vector(length=expression_length)
  indices[seq(1, expression_length, 2)] <- number_indices
  indices[seq(2, expression_length, 2)] <- operator_indices

  return(indices)
}
```

Here is a sample agent:

```
agent <- generate_agent()
agent
```

```
## [1] 2 9 5 8 4 6 1 8 3
```

This function generates a random population by repetitive calling of `generate_agent` .

```r
generate_population <- function(object) {
  indices <- matrix(NA, nrow = object@popSize, ncol = expression_length)
  for (i in 1:object@popSize) {
    indices[i, ] <- generate_agent()
  }

  return(indices)
}
```

Fitness function

Fitness function evaluates the expression and compares the result with `target_value`.

```r
my_fitness <- function(agent) {
  expression <- numbers_and_operators[agent]
  value <- eval(parse(text = paste(expression, collapse = "")))
  return(-abs(value - target_value))
}

my_fitness_2 <- function(agent) {
  expression <- numbers_and_operators[agent]
  value <- eval(parse(text = paste(expression, collapse = "")))
  return(1/abs((value - target_value)))
}
```

Print agent function

```r
print_agent <- function(agent){
  expression = paste(numbers_and_operators[agent], collapse = "")
  value = eval(parse(text = expression))
  fitness = my_fitness(agent)
  print(paste(
    c(
      "[", paste(agent, collapse = ", "), "]",
      "; ", expression, " = ", value, "; fitness = ", fitness
    ), collapse = ""
  ))
}

print_agent(agent)
```

```
## [1] "[2, 9, 5, 8, 4, 6, 1, 8, 3]; 25*3/5+10/100 = 15.1; fitness = -2496.9"
```

Mutation function

Mutation my swap two numbers or two operators.

```r
my_mutation <- function(object, parent) {
  mutate <- parent <- as.vector(object@population[parent, ])
  n <- n_numbers * 2 - 1
  rand <- round(runif(1, 0, 1)) # rand: 0 = number, 1 = operator
  swap <- seq(1 + rand, n, 2)   # 0 -> (1 3 5 7 9)    1 -> (2 4 6 8)
  # swap 2 numbers / operators
  indexes <- sample(swap, 2)
  mutate[indexes[1]] <- parent[indexes[2]]
  mutate[indexes[2]] <- parent[indexes[1]]

  return(mutate)
}
```

Crossover function

Crossover takes two parents and produces two children. Each agent has 4 spots for operators and 5 spots for numbers.

Operator crossover Firstly some random operator spots are chosen and stored in the variable `switch_indices`. Child one gets operators on this spots from first parent and other operators from second parent. Child two gets operators on this spots from second parent and others from first.

```
my_crossover <- function(object, parent) {
  pop <- object@population
  n <- n_numbers * 2 - 1
  p1 <- pop[parent[1], ]
  p2 <- pop[parent[2], ]
  children <- matrix(NA, nrow = 2, ncol = n)
  children[1, ] <- p1
  children[2, ] <- p2

  ssize <- round(runif(1, 1, n_numbers - 1))
  operator_indices <- sample(seq(2, n, 2), ssize)

  children[1, operator_indices] <- p2[operator_indices]
  children[2, operator_indices] <- p1[operator_indices]

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

Number and operator crossover Expressions are treated like a permutation.

Some consecutive random spots that begin with a number and end with symbol are chosen and stored in a variable `indices`. Child one gets the beginning segment of the expression from parent one. Numbers that not yet present are copied from parent two together with the symbol right after them.

For example:

$$
\begin{aligned}
\text{Parent 1:} &\quad 5 \,/\, \underbrace{100 + 25 -}_{selected} \; 3 \cdot 10 \\
\text{Parent 2:} &\quad \underbrace{3 +}_{remaining} \quad 25 + \underbrace{10 + 5 -}_{remaining} \; 100 \\
\text{Child 1:} &\quad \underbrace{100 + 25 -}_{P1}\underbrace{3 + 10 + 5}_{P2}
\end{aligned}
$$

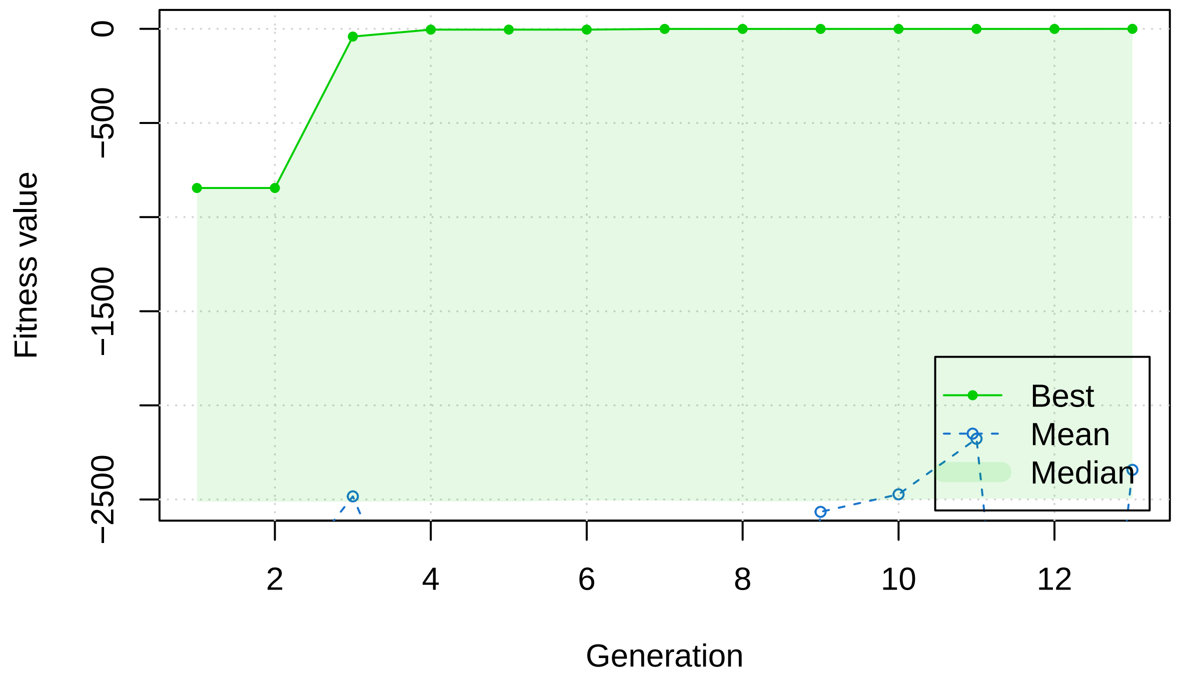
```
GA <- ga(type = "permutation", population = generate_population,
         fitness = my_fitness,
         mutation = my_mutation,
         crossover = my_crossover,
         maxFitness = 0,
         maxiter = 500,
         popSize = 100, lower = 1, upper = 9)
summary(GA)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  100
## Number of generations =  500
## Elitism               =  5
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 13
## Fitness function value = 0
## Solution =
##      x1 x2 x3 x4 x5 x6 x7 x8 x9
## [1,]  1  6  4  7  5  6  3  9  2
```

```
for (i in 1:nrow(GA@solution)) {
  a <- c(t(matrix(GA@solution[i, ])))
  print_agent(a)
}
```

```
## [1] "[1, 6, 4, 7, 5, 6, 3, 9, 2]; 10+5-3+100*25 = 2512; fitness = 0"
```

```
plot(GA)
```



Random search

This function just generates a random expression, checks if it has desired value and repeats.

```
random_search <- function(){
  found = FALSE
  counter = 0
  while(!found){
    a = generate_agent()
    counter <- counter + 1
    if(my_fitness(a) == 0){
      found = TRUE
    }
  }
  return(list(agent=a, iterations=counter))
}

result = random_search()
print_agent(result$agent)
```

```
## [1] "[1, 6, 2, 9, 3, 6, 4, 7, 5]; 10+25*100+5-3 = 2512; fitness = 0"
```

```
print(paste(c("Solution found in ", result$iterations, " iterations."), collapse = ""))
```

```
## [1] "Solution found in 2688 iterations."
```

Let’s see how many iterations does it take, on average, to guess the correct expression.

```r
all_iterations = 0
N = 100
for (i in 1:N) {
  result <- random_search()
  all_iterations <- all_iterations + result$iterations
}
print(paste(c("On average solution was found in ", all_iterations/N, " iterations."), collapse = ""))
```

```
## [1] "On average solution was found in 2368.59 iterations."
```