# Intelligent Systems Seminar Assignment 1

November 1, 2021

## 1 A general note

This is the first programming assignment for the Intelligent Systems labs. Your final output should be a markdown file, rendered as a PDF (see https://rmarkdown.rstudio.com/ for more info). You will be asked to provide such rendered markdown files (PDFs) as your solutions; you are allowed to solve the task either individually or in pairs. If you are using any other language than the one used during labs (R), please make sure that the presentation of your work is of comparable quality (markdown with documented steps + intermediary results/plots etc.). **Good luck!**

## 2 Problem definition

The goal of the first seminar assignment is to solve a simple mathematical game using genetic algorithms (examples are given in R). In this game, you are given a set of numbers, a set of mathematical operators, and a target number. Your goal is to arrange the numbers and mathematical operators in such a way that the result of the equation comes as close to the target number as possible. For example:

- You are given the numbers $10, 25, 100, 5, 3$ and operators $+, -, /, *$
- The target number is 2512

A solution that matches the target number would be $100 * 25 + 10 + 5 - 3 = 2512$. Depending on the set of numbers, reaching the target number may not always be possible.

To make the problem slightly easier, the following rules also apply:

- Each number can only be used once
- Each operator can be used multiple times
- Your solutions should not include brackets

## 3 Tasks

### 3.1 Population (25%)

In your first task, think about how you can represent your solutions in a format that is compatible with the R GA library. Keep in mind that the GA library can work with three different types of vectors (vectors of real-valued values, vectors of binary values, and permutations). Pick the type most suited for this assignment and create a function that will return starting population.

Note that the GA library does not work with permutations of strings, therefore you can't represent your solutions as string vectors. However, you can represent them as indices of a string vector. For example:

```
# This kind of vector will not work with the GA library
c("100", "*", "25", "+", "10", "+", "5", "-", "3")

# This will work with the GA library
numbers_and_operators <- c("10", "25", "100", "5", "3", "+", "-", "/", "*")
```

```
x = c(3, 9, 2, 6, 1, 6, 4, 7, 5)
numbers_operators[x]
```

If you use your own function to generate a starting population, a permutation can include the same number multiple times.

## 3.2 Fitness function (10%)

Implement a fitness function that can be used to optimize this problem. Hint: if you have a string vector of a mathematical expression, you can obtain the result of the expression in the following way:

```
> a = c("1", "+", "2")
> str = paste(a, collapse='')
> eval(parse(text=str))
[1] 3
```

## 3.3 Crossover and mutation functions (40%)

The crossover and mutation functions that come with the GA library are not well-suited for this problem. For example, the swap mutation function that comes with GA will swap two random elements without ensuring that the result is a valid equation. Write your own functions which will perform mutation and crossover in such a way that they generate valid equations.

You can base your crossover and mutation functions on existing GA library functions (https://github.com/luca-scr/GA/blob/master/R/genope.R). Modify one crossover or mutation function in such a way that it returns valid equations.

For example, GA implements the swap mutation using the following function:

```
gaperm_swMutation_R <- function(object, parent)
{
  # Select a parent from the population
  mutate <- parent <- as.vector(object@population[parent,])
  n <- length(parent)
  # Sample two random numbers from 1 to length(parent)
  m <- sample(1:n, size = 2)
  # Swap the elements in the selected numbers
  mutate[m[1]] <- parent[m[2]]
  mutate[m[2]] <- parent[m[1]]
  return(mutate)
}
```

The function takes two paramters:

1. **object**, which is the object of class GA, the same class returned as the result of the ga function,
2. **parent**, which is a number telling the function which object in the population was selected for mutation.

The function returns the vector mutate, which represents the modified element.

Crossover functions are implemented in a similar way, with two major differences:

1. The parameter parents now has two (or more) numbers,
2. The function now returns a list in the following format: `list(children = children, fitness = fitness)`, where children are the newly generated elements and fitness are their fitness values, which can either be computed in the mutation function or set as NAs ( `fitness = rep(NA, 2)`).

As long as your functions follow the same format, you can use them in the ga function by using the mutation/crossover parameters.

### 3.4 Evaluation (25%)

Evaluate your solution by comparing it to a random search. First, implement a random search function that attempts to find the best solution by generating random equations. Then compare the time taken by the random search to the time taken by your genetic algorithm. Perform this evaluation on multiple test cases with varying amounts of available numbers and present the results with a graph. Additionally, try using multiple selection/mutation/crossover functions and other parameters to determine what works best for this specific problem. Discuss the impact of using different evolution configurations on the search.

### 3.5 Bonus task (10%)

Modify your approach so that each number can be used up to five times. Then, instead of finding the closest solution to the target number, try to find the **shortest** such solution. Modify the population, mutation, and crossover, and fitness functions as necessary and then evaluate your approach in the same manner as in the previous task.

## 4 A note on using other programming languages

If you prefer, you can solve the seminar assignment using other programming languages. However, you'll still need to program the required tasks by yourself. That is, even if you use a library in another programming language that already has some of the above tasks implemented, you should still code them yourselves. Similarly, you can solve the task in R without using the GA library (for example, by programming a simple genetic algorithm yourselves).