EVOLUTIONARY COMPUTATION

MAKE-UP

2020 - 2021

Submitted by

Halil Ibrahim Akboğa

160709024

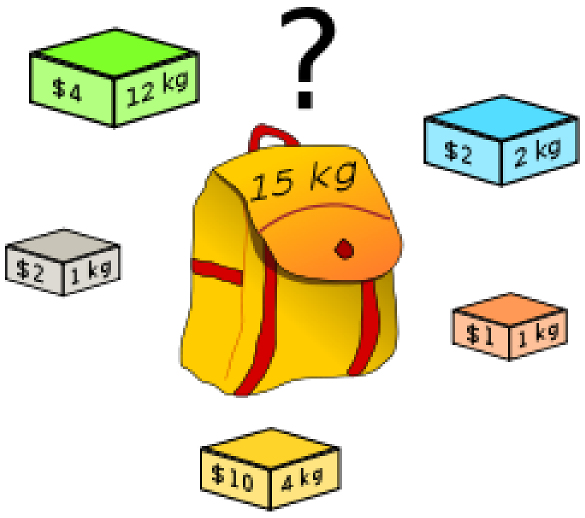
Topic

Solution for Knapsack Problem using with Genetic Algorithm

(R Language)

Introduction / What is Knapsack Problem?

In this project, we are going to make an genetic algorithm for the Knapsack problem with R language. But first, we need to know or problem. The knapsack problem is a problem in combinational optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively. The knapsack problem has been studied for more than a century, with early works dating as far back as 1897. The name "Knapsack Problem" dates back to the early works of the mathematician Tobias Dantzig (1884-1956) and refers to the commoplace problem of packing the most valuable or useful items without overloading the luggage.



original image from wikipedia

Methodology

Now, let's look at our methodology for this solution, the knapsack problem that we are trying to solve is 0-1 knapsack problem. Generally people works this topic on dynamic programming. Given a set of n items numbered from 1 to n, each with weight w_i and a value v_i. Suppose that each item copies are restricted to 1, i.e. the item is either included in the knapsack or not. Here we want to maximize the objective function (i.e. the values of items in the knapsack).

$$\sum_{i=1}^{n} v_i . x_i$$

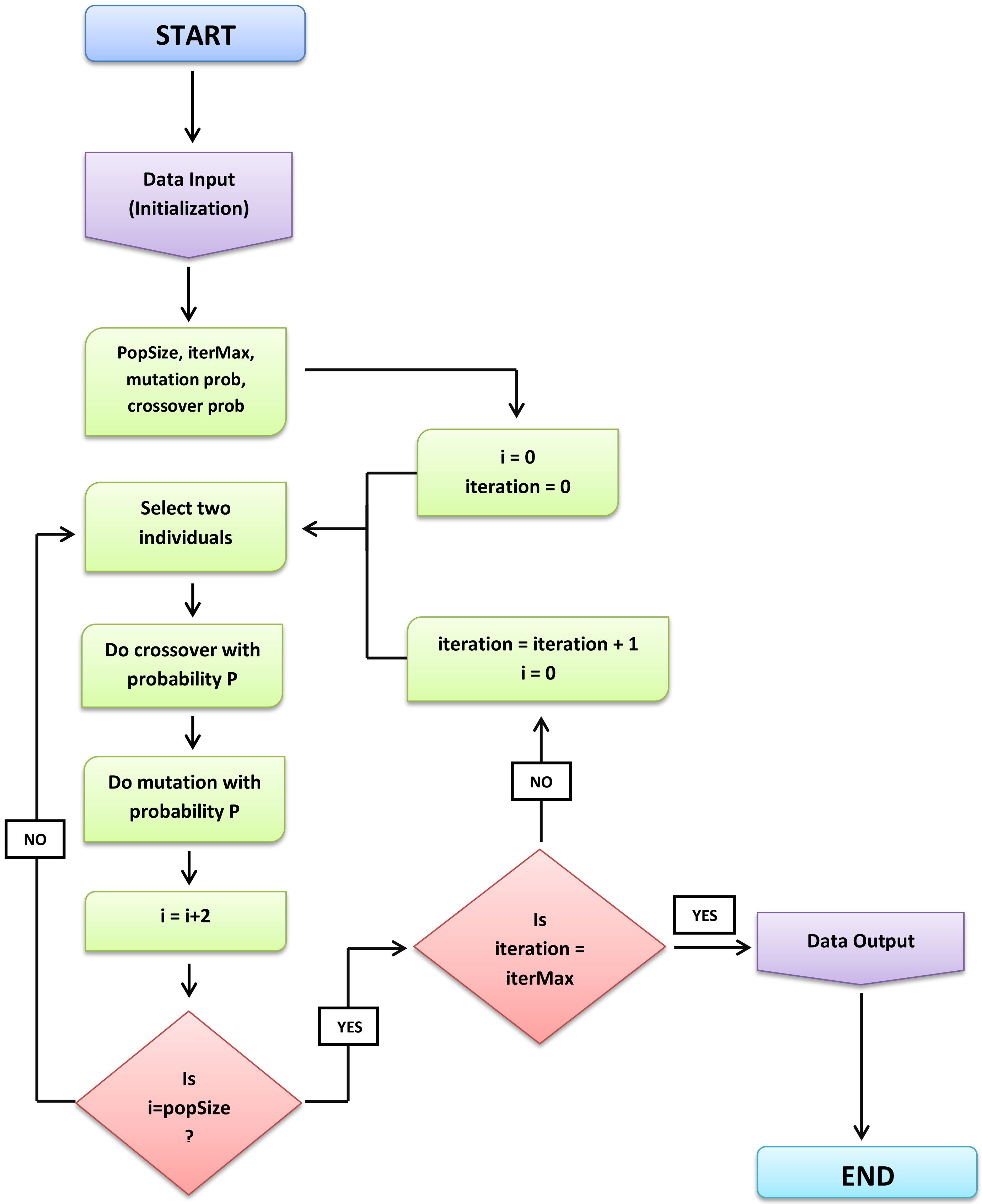
where the objective function is subject to the constraint function

$$\sum_{i=1}^{n} w_i . x_i$$

x_i between {0,1}

Genetic Algorithm Methodology

Genetic algorithm is one of the optimization algorithms based on the evolution concept by natural selection. As proposed by Charles Darwin, evolution by natural selection is the mechanism on how many varieties of living things will adapt to the environment to survive through two principle: natural selection and mutation. Based on that concept, the Genetic Algorithms goal is to gain the optimal solutions of the objective function by selecting the best or fittest solution alongside the rare and random mutation occurrence. The algorithm itself can be explained as the next page scratch.


START
Data Input
(Initialization)
PopSize, iterMax,
mutation prob,
crossover prob
i = 0
iteration = 0
Select two
individuals
Do crossover with
probability P
Do mutation with
probability P
i = i+2
Is
i=popSize
?
NO
YES
Is
iteration =
iterMax
NO
iteration = iteration + 1
i = 0
YES
Data Output
END

- Initialize the data and/or the function that we will optimize.
- Initialize the population size, maximum iteration number (the number of generations), crossover probability, mutation probability and the number of elitism (the best or fittest individual that will not undergo mutation).
- From the population, select two individuals, then perform crossover between two individuals with the probability p.
- Then, perform mutation between two individuals with probability p (usually the probabilty of mutation are really low).
- Repeat steps 3 and 4 until all individuals in one generation are trained. These all individuals will be used for training the next generation until the number of generations is reaching the limit.

If we see the steps clearly, let’s continue with R part but first we should imagine a problem.

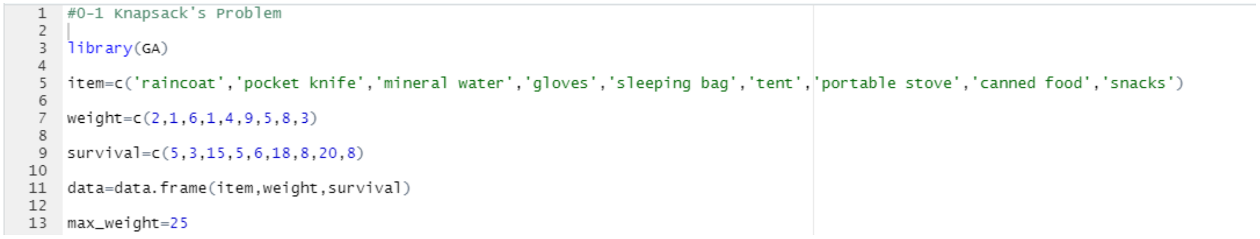
Solving Knapsack Problem using Genetic Algorithm

Imagine that you want to go hiking with your family and you have the items that you can use for hiking, with the weight (kilogram) and survival point of each item, respectively, as follows;

ITEMS	**WEIGHT**	**SURVIVAL POINT**
Snacks	3	8
Canned Food	8	20
Portable Stove	5	18
Tent	9	18
Sleeping Bag	4	6
Gloves	1	5
Mineral Water	6	15
Pocket Knife	1	3
Raincoat	2	5

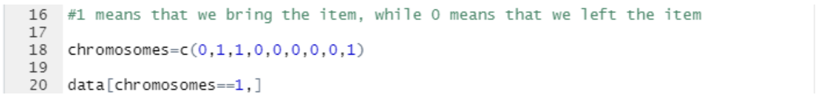
Suppose too that, you have a knapsack that can contain items with a maximum capacity of 25 kilograms, where you can only bring one copy for each item. The goal is that you want to maximize your knapsack capacity while maximizing your survival points as well. From the problem statement, we can define the weight of the items as the constraint function, while the survival points cumulated from the items in the knapsack as the objective function.

Now, we are getting to coding part; for implementation of this project, we are going to use GA libtary in R. First, we need to input the data and the parameters that we used by writing codes.



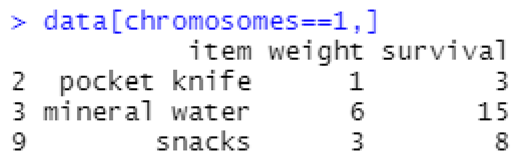
```r
#0-1 Knapsack's Problem

library(GA)

item=c('raincoat','pocket knife','mineral water','gloves','sleeping bag','tent','portable stove','canned food','snacks')

weight=c(2,1,6,1,4,9,5,8,3)

survival=c(5,3,15,5,6,18,8,20,8)

data=data.frame(item,weight,survival)

max_weight=25
```

To give you a better understanding of the genetic algorithm in this problem, suppose that we only bring a pocket knife, mineral water and snacks in our knapsack, initially. We can write it as the “chromosome” and since the problem that we want to solve is a 0-1 knapsack problem, then from these lines of codes below, 1 means that we bring the item, while 0 means that we left the item.



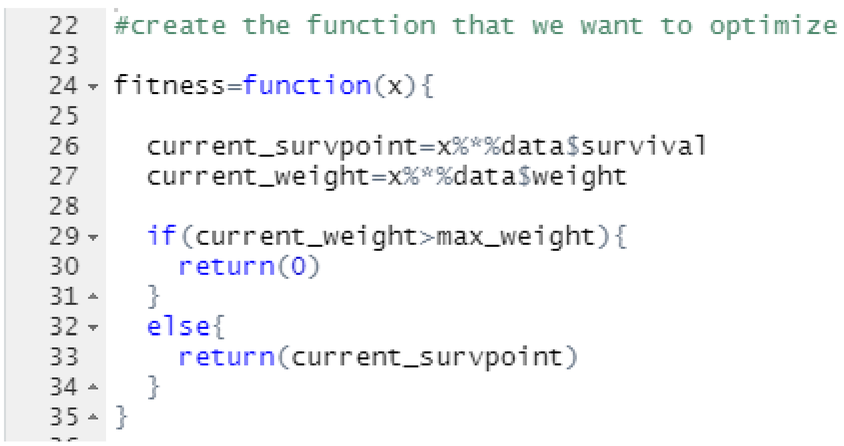
```r
#1 means that we bring the item, while 0 means that we left the item

chromosomes=c(0,1,1,0,0,0,0,0,1)

data[chromosomes==1,]
```

If we run the codes, we will get this output:



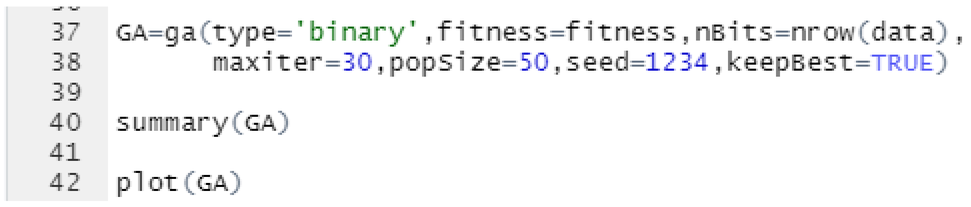
```
> data[chromosomes==1,]
           item weight survival
2  pocket knife      1        3
3 mineral water      6       15
9        snacks      3        8
```

Then, we create the objective function that we want to optimize with the constraint function by writing these lines of code below. The *fitness* the function will we use in the ga function from the GA library.



```r
#create the function that we want to optimize

fitness=function(x){

  current_survpoint=x%*%data$survival
  current_weight=x%*%data$weight

  if(current_weight>max_weight){
    return(0)
  }
  else{
    return(current_survpoint)
  }
}
```

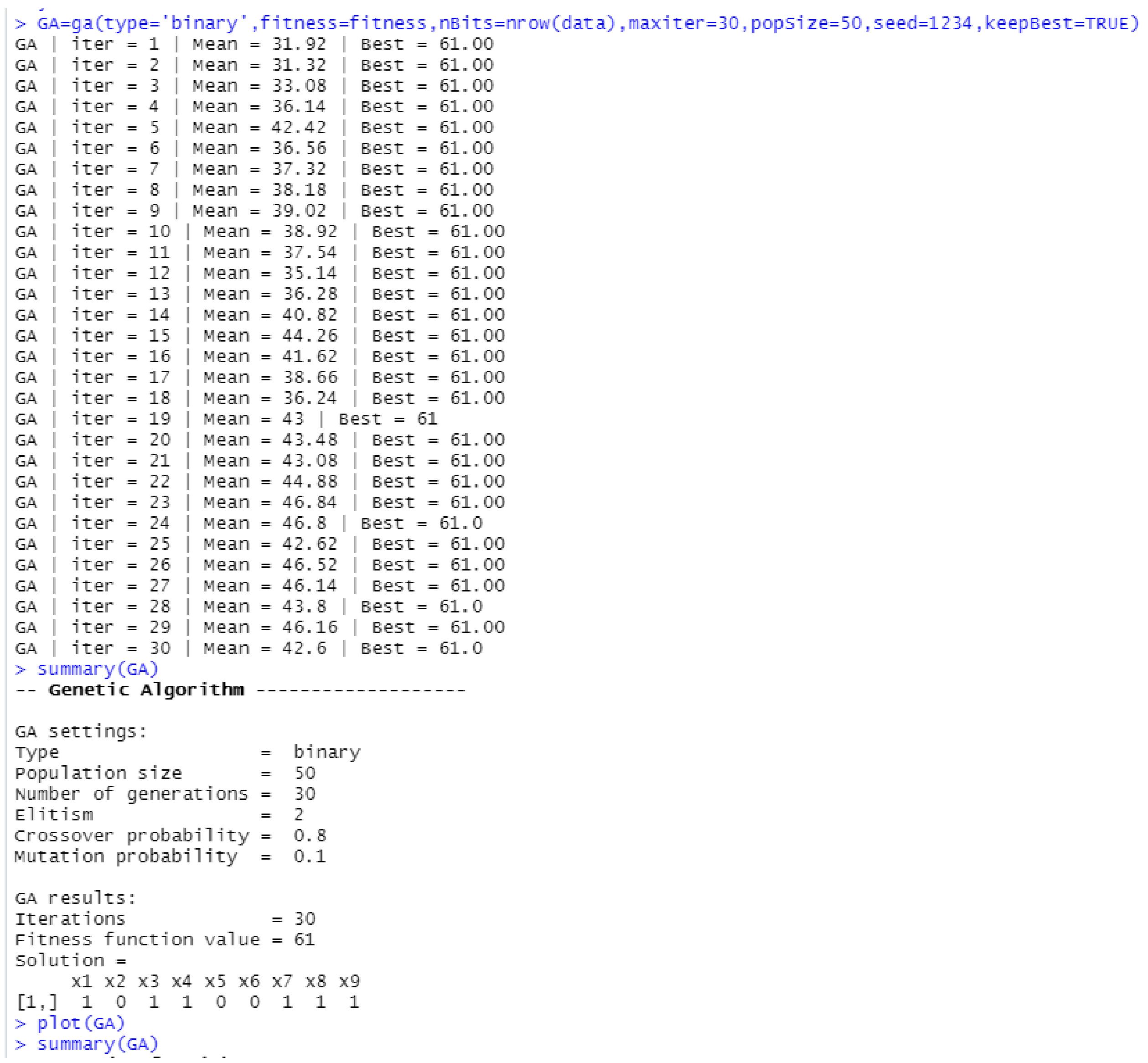
Now, here is the interesting part: the optimization process using the genetic algorithm. Suppose that we want to create a maximum of 30 generations and 50 individuals for the optimization process. For reproducibility, we write the *seed* argument and keep the best solution.



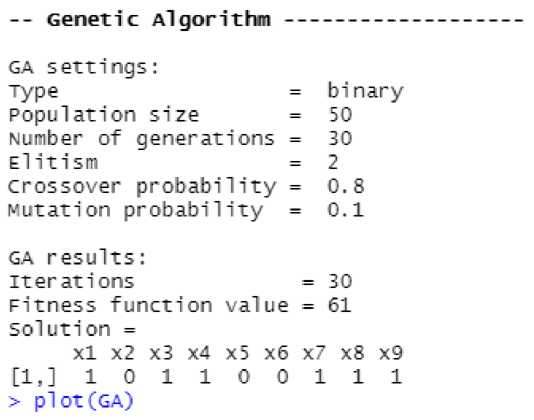
```r
GA=ga(type='binary',fitness=fitness,nBits=nrow(data),
      maxiter=30,popSize=50,seed=1234,keepBest=TRUE)

summary(GA)

plot(GA)
```

If we run the codes, output will be like that in next page;

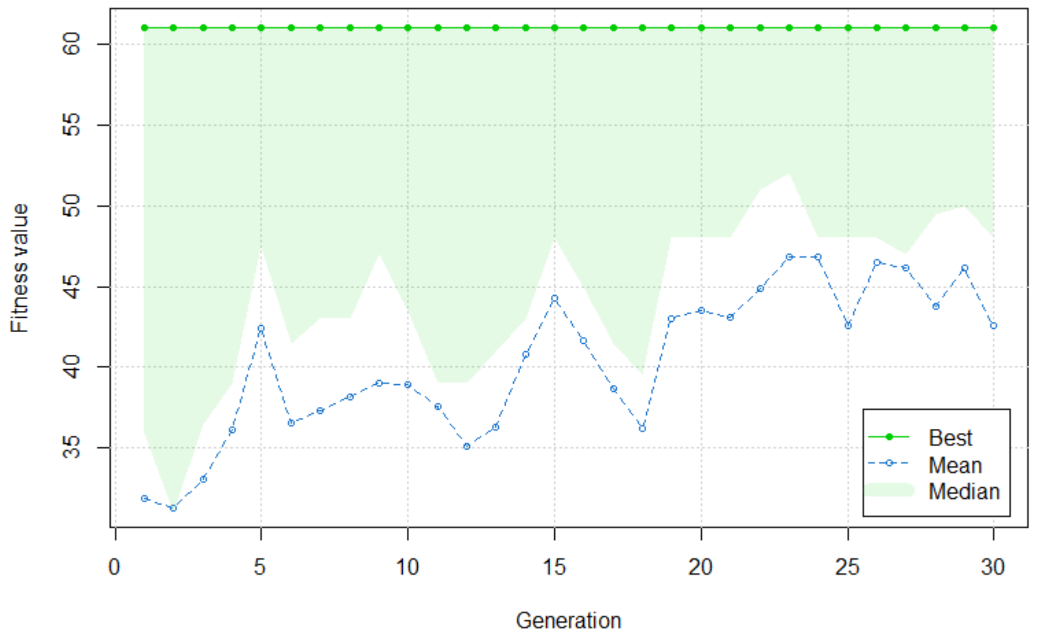
Output;



```
> GA=ga(type='binary',fitness=fitness,nBits=nrow(data),maxiter=30,popSize=50,seed=1234,keepBest=TRUE)
GA | iter = 1 | Mean = 31.92 | Best = 61.00
GA | iter = 2 | Mean = 31.32 | Best = 61.00
GA | iter = 3 | Mean = 33.08 | Best = 61.00
GA | iter = 4 | Mean = 36.14 | Best = 61.00
GA | iter = 5 | Mean = 42.42 | Best = 61.00
GA | iter = 6 | Mean = 36.56 | Best = 61.00
GA | iter = 7 | Mean = 37.32 | Best = 61.00
GA | iter = 8 | Mean = 38.18 | Best = 61.00
GA | iter = 9 | Mean = 39.02 | Best = 61.00
GA | iter = 10 | Mean = 38.92 | Best = 61.00
GA | iter = 11 | Mean = 37.54 | Best = 61.00
GA | iter = 12 | Mean = 35.14 | Best = 61.00
GA | iter = 13 | Mean = 36.28 | Best = 61.00
GA | iter = 14 | Mean = 40.82 | Best = 61.00
GA | iter = 15 | Mean = 44.26 | Best = 61.00
GA | iter = 16 | Mean = 41.62 | Best = 61.00
GA | iter = 17 | Mean = 38.66 | Best = 61.00
GA | iter = 18 | Mean = 36.24 | Best = 61.00
GA | iter = 19 | Mean = 43 | Best = 61
GA | iter = 20 | Mean = 43.48 | Best = 61.00
GA | iter = 21 | Mean = 43.08 | Best = 61.00
GA | iter = 22 | Mean = 44.88 | Best = 61.00
GA | iter = 23 | Mean = 46.84 | Best = 61.00
GA | iter = 24 | Mean = 46.8 | Best = 61.0
GA | iter = 25 | Mean = 42.62 | Best = 61.00
GA | iter = 26 | Mean = 46.52 | Best = 61.00
GA | iter = 27 | Mean = 46.14 | Best = 61.00
GA | iter = 28 | Mean = 43.8 | Best = 61.0
GA | iter = 29 | Mean = 46.16 | Best = 61.00
GA | iter = 30 | Mean = 42.6 | Best = 61.0
> summary(GA)
-- Genetic Algorithm -------------------

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  30
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 30
Fitness function value = 61
Solution =
     x1 x2 x3 x4 x5 x6 x7 x8 x9
[1,]  1  0  1  1  0  0  1  1  1
> plot(GA)
> summary(GA)
```



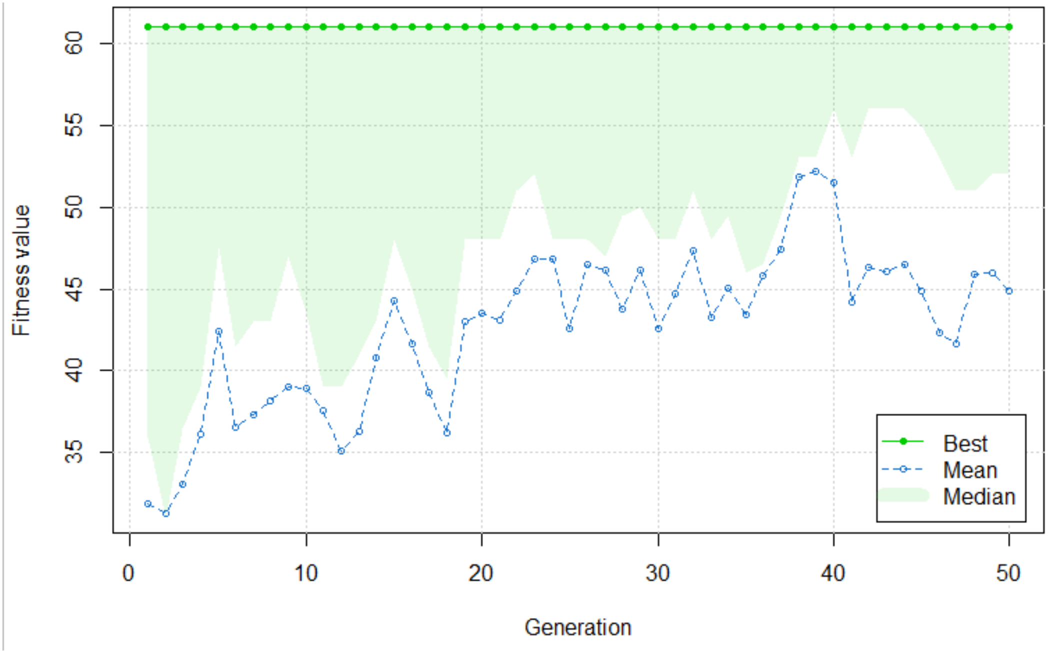
```
-- Genetic Algorithm -------------------

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  30
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 30
Fitness function value = 61
Solution =
     x1 x2 x3 x4 x5 x6 x7 x8 x9
[1,]  1  0  1  1  0  0  1  1  1
> plot(GA)
```



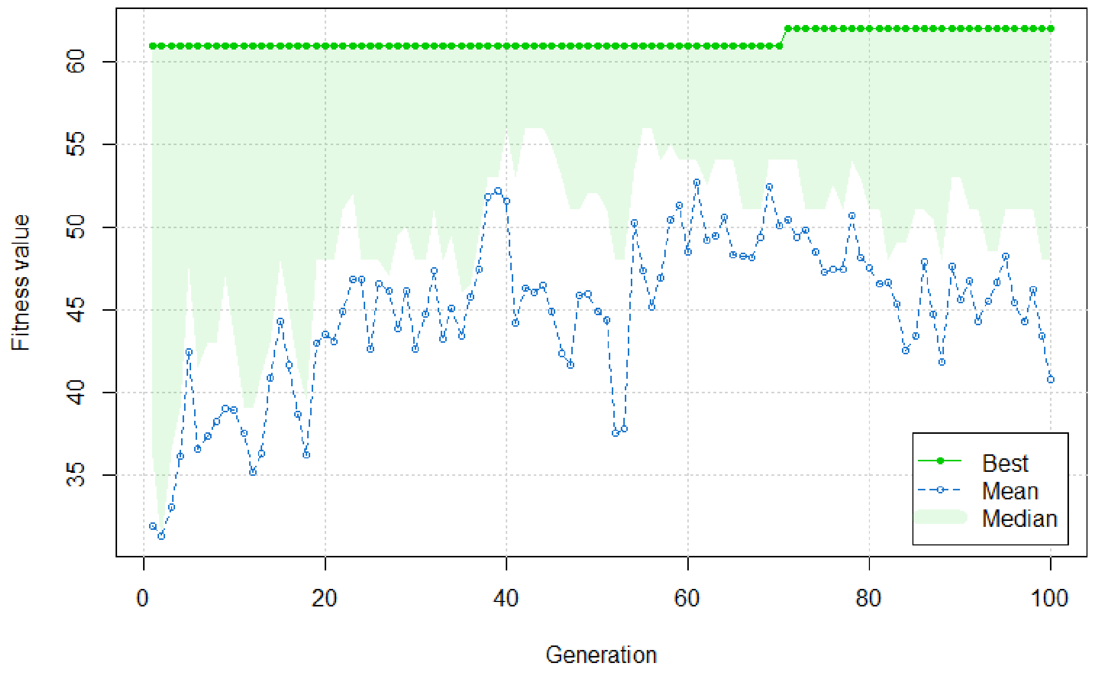
The Genetic Algorithm Optimization Test

From the results above, we can see that the performance for every individual is increasing in each generation. We can see it from the fitness value mean and median that tends to increase in each generation. Let’s try to train it again, but with more generations.

```
GA2=ga(type='binary',fitness=fitness,nBits=nrow(data),maxiter=50,popSize=50,seed=1234,keepBest=TRUE)

GA3=ga(type='binary',fitness=fitness,nBits=nrow(data),maxiter=100,popSize=50,seed=1234,keepBest=TRUE)

plot(GA2)

plot(GA3)
```

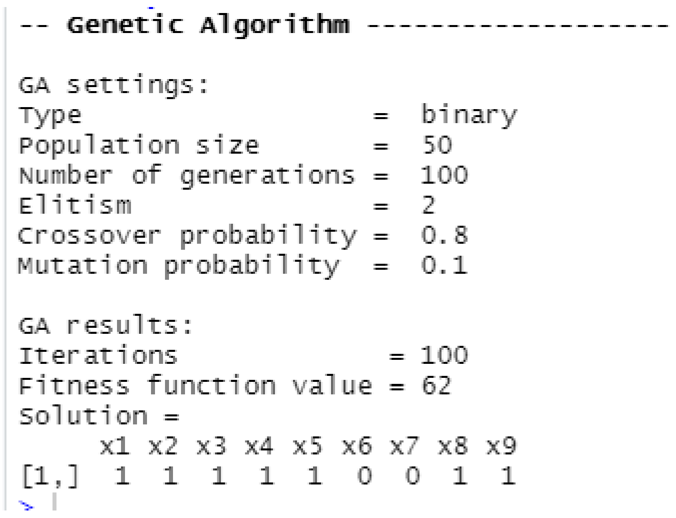


The Genetic Algorithm Optimization Test –GA2

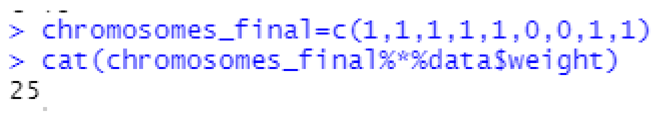


The Genetic Algorithm Optimization Test –GA3

From GA2 and GA3, we can see that the optimization result for each individual is at their best on generation 40-ish and 60-ish, according to the mean and median of fitness value on that generation. We can also see that the best fitness value is increasing to 62 from 72nd generation onwords. Since we keep the best result on every optimization process, we want to find out the items that we can bring for hiking based on the best result from the genetic algorithm optimization. We can see the summary from GA3 as follows.


```
-- Genetic Algorithm -------------------

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  100
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 100
Fitness function value = 62
Solution =
     x1 x2 x3 x4 x5 x6 x7 x8 x9
[1,]  1  1  1  1  1  0  0  1  1
>
```

From the result above, we can conclude that the items that we can include in the knapsack are a raincoat, pocket knife, mineral water, gloves, sleeping bag, canned food and snacks. We can calculate the weight of the knapsack, to make sure that the knapsack is not over capacitated.



```
> chromosomes_final=c(1,1,1,1,1,0,0,1,1)
> cat(chromosomes_final%*%data$weight)
25
```

As you see, the weight of the items is the same as the knapsack capacity!

Conclusion / Result

And you are ready to go! You can go to hiking with your family or friends with maximum survival points and capacity by implementing the genetic algorithm in R. In fact, you can solve the knapsack problem by using genetic algorithm in many real-world problems applications, such as choosing the best performing portfolio, production scheduling and many more.

Thank you.

References

https://en.wikipedia.org/wiki/Knapsack_problem

https://cran.r-project.org/web/packages/GA/GA.pdf

https://rpubs.com/Argaadya/550805

https://blog.datascienceheroes.com/feature-selection-using-genetic-algorithms-in-r/