Project: Exploration of Genetic Algorithms

Anurag Maji
M10726841

Introduction

This project deals with the explanation of various facets and applications of Genetic Algorithms, which is a search procedure to compute true or approximate solutions to optimization and search problems. They belong to a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

Executive Summary

This project explains the steps and components (along with options available) of the Genetic Algorithm with one iteration of a maximization problem. Further, a brief overview of the applications of GA with Data Mining have been mentioned. A comparative study of the practical application of the methodology adopted by Genetic Algorithm is made against Simulated Annealing using the Traveling Salesman Example. A functional implementation of Genetic Algorithms was performed in R to find the global minima.

Genetic Algorithms

This project explores Genetic Algorithms, a search procedure to find good solution to an optimization problem where traditional algorithms begin to fail or do not perform as desired. It is a part of evolutionary algorithms and is inspired by the natural selection process in evolution of species. This algorithm performs random sampling from the population and gives out results that are used for moving towards the next step. The 'optimal' solution is achieved over successive steps. In short, this process is iterative in nature and by continuously correcting itself by using previous information, it arrives at a good solution. Something to be noted is that it can not be claimed that a solution obtained from GA is an optimal solution since it is often not possible to determine what the real optimum is.

This algorithm can be used for problems where the objective function is highly non linear. Also, it is more robust to changes in general when compared to conventional algorithms. These algorithms perform well when limited amount of information is available. Other methods to find suitable solutions include hill climbing, tabu search and simulated annealing.

Key Principles in Genetic Algorithms

Key principles in Genetic Algorithms are derived from Darwinian Natural Selection.

Heredity: There must be a process in place by which children receive the properties of their parents.

Variation: There must be a variety of traits present in the population or a means with which to introduce variation.

Selection: There must be a mechanism by which some members of a population have the opportunity to be parents and have the opportunity to pass down their genetic information and some do not. Based on the mechanism “Survival of the fittest”.

Operators for Genetic Algorithms

Selection : This refers to the selection procedure for the parents from the population for crossover. There are several methods available –

- Roulette wheel Selection : Fitness for each parent is calculated and probability for selection is proportional to the fitness. Chromosomes with greater fitness are selected more number of times. But if the fitness differs by much, some parents would be selected almost always and some very rarely.
- Rank Selection : Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The fitness rankings are in ascending order with the worst will have fitness ***1***, second worst ***2*** and so on. The best will have fitness ***N*** (number of chromosomes in population). The chromosomes are then chosen accordingly. But this might lead to slower convergence since all chromosomes have a decent chance of getting selected.

Encoding : This refers to the representation of information and solution within the GA. Common ways of encoding information include –

- Binary : Each chromosome is a binary string, with each bit representing some information.
- Permutation Encoding: Every chromosome is a string of numbers, which are represented in a sequence.
- Value Encoding : It involves a string of values .
- Tree Encoding: This involves representing a chromosome as a tree of objects .

The type of encoding would depend on the problem at hand. For example –Binary encoding is suitable for the Knapsack Problem where 0 and 1 would indicate if an item is in the Knapsack. The Traveling Salesman Problem would use Permutation Encoding where the string would represent the order of cities. Value encoding is suitable for assigning numerical values – like weights in a neural network. Tree encoding can be used to define functions that could relate a set of input and output values.

Crossover : It is the process of crossing two members of the existing population to create two new members for the population by means of encoding some information from each parent. If no crossover is performed, offspring would be an exact copy of their parents. Offsprings are generally retained only if they are an improvement over the least good individual in the previous population. Crossover could be at a single point, at two points, uniformly distributed or based on a Boolean operator (AND).

Another interesting variation includes Selective crossover where one of the offspring gets all the dominant characteristics from either parents and the other gets the alternatives. If a single child's fitness is greater than the fitness of either parent, the dominance values (of those genes that were exchanged during crossover) are increased proportionately to the fitness increase. This is done to reflect the genes' contribution to the fitness increase.

Crossover Probability : This defines how often the crossover takes place between parents and offsprings. Crossover Probability = 100% means that the crossover takes place for each every offspring and crossover probability = 0 % means that offsprings are exact copies of their parents. An intermediate value should be chosen to retain some parts of the previous generation. As per study, Crossover rate generally should be high, about **80%-95%.**

Mutation: It is the stage where a chromosome randomly mutates to produce variation. It is intended to occasionally break one or more members of a population out of a local minimum/maximum space and potentially discover a better minimum/maximum space. But it can go either way where sometimes the mutations stimulate a population that moves toward the goal in leaps and bounds, other times, the mutation slow road in wrong direction.

Mutation Probability : This defines how often there are random changes within the offspring. 0% mutation means there is no mutation after the crossover. 100% mutation probability means everything is changed randomly after the crossover. As we increase the percentage of mutation, the Genetic Algorithm moves towards becoming random search and hence an intermediate value is preferable.

Population size : This defines the number of chromosomes in one population i.e. in one generator. A very small population might mean less crossover possibility and hence smaller exploration of the search space. A large population might mean large possibilities of crossover but might slow down the algorithm due to the number of choices available.

A common theme that comes out in Genetic Algorithm is **Eliticism**. This is important to ensure that the best found solution is retained. In this process, the best found solution is first retained and the rest of the process is done in a classical way.

<u>Basic Steps in GA</u>

The similarity of steps followed in GA to natural selection can be noted, since it follows the principle of survival of the fittest.

Step 1 - The algorithm selects certain entities randomly at the initial stage to form the original population.

Step 2 – The 'fitness' is evaluated for each member of the population by a certain performance criteria that can be set by us. Hence, if an entity performs better as per the criteria assigned i.e. it has a higher fitness score, more are its chances of getting selected.

Step 3 - The next step is crossover between selected individuals to produce new entities or 'offspring'. Several modifications can be made on how the process can be performed, but it is analogous to the production of an offspring in the biological world where the offspring inherits some properties from both parents.

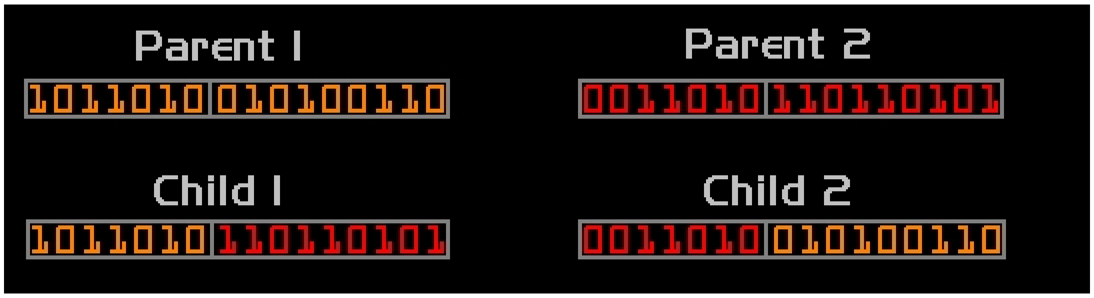


Fig 1: Parent - Offspring

Step 4 – This step involves mutation, which is to replace some of the characteristics of the entities generated so as to ensure diversity in the new population(crowd reduction). Alternatives to this include reduction of fitness of an entity if the population already contains many similar individuals to it. This is mainly done to introduce variation in the population. It is important to note that this is performed only on a small proportion of the new population.

The survivors are selected again and the process loops over till a suitable solution is achieved.

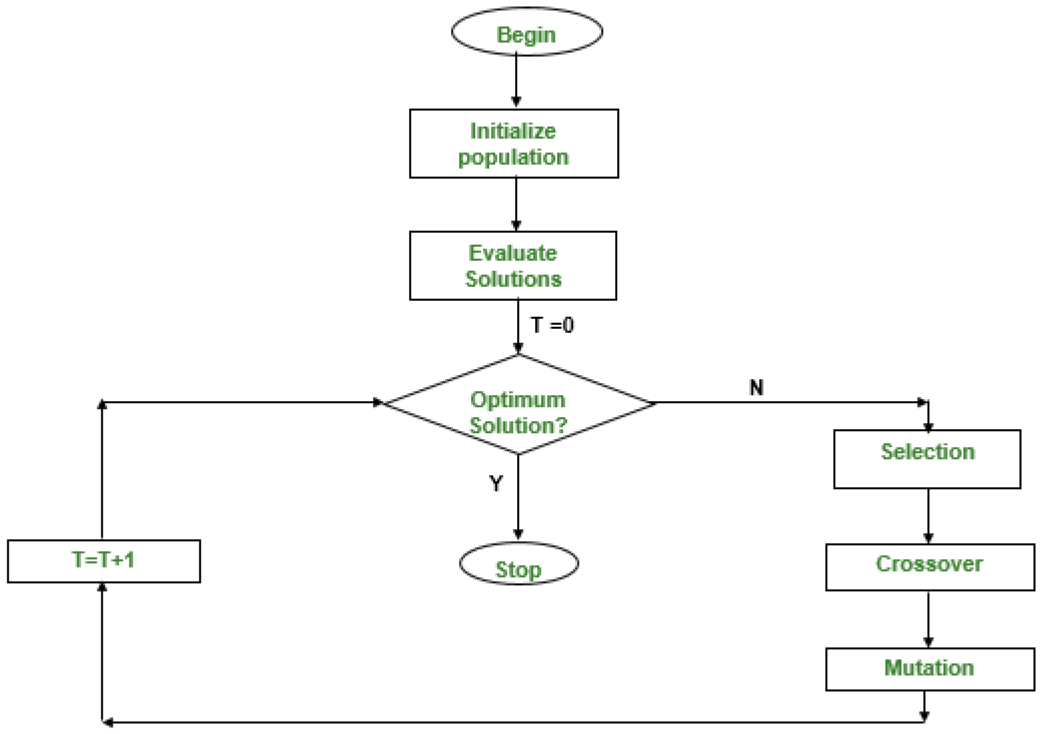


Fig 2: Steps in GA

Sample application in Optimizing a function

Consider a simple example of maximizing the function $x^2 + 5x + 100$ where x is an integer between (0,30). We can see that the function is non linear and may result in local optima if the search space becomes more complicated. The encoding used to represent solutions in Genetic Algorithms is binary. The explanation below provides one iteration of the algorithm.

First step is to randomly select a population in the search space. The value of f(x) is the fitness value. The probability of selection is proportional to the fitness score here.

	16	8	4	2	1	x		f(x)	P(x)
1	0	1	0	0	0	8		204	0.088696
2	1	0	1	0	0	20		600	0.26087
3	0	1	1	0	0	12		304	0.132174
4	0	0	1	0	1	5		150	0.065217
5	1	1	0	1	0	26		906	0.393913
6	0	0	1	0	0	4		136	0.05913
							Average	383.3333	

Table 1: Sample iteration in Excel

The table above shows 5 randomly chosen members, their function value and associated probabilities. Now, we select two individuals randomly for reproduction. Say individual 1 and 5 are chosen. Crossover is performed on these individuals for which we again randomly pick a point on the string for a crossover, say point 3 in this case. Two offsprings would be produced in this case.

Mating Pair	New Pair
01000	01010
11010	11000

Table 2: Offsprings

The offsprings produced are as follows with the given

0	1	0	1	0		10		250
1	1	0	0	0		24		796

Table 3: Offspring Coding

Now these two offsprings are to be added to the population, but the population size needs to be maintained. Hence two members from the original population would be eliminated. Say 1 and 6 are eliminated and replaced by the offsprings.

	16	**8**	**4**	**2**	**1**		x		f(x)	P(x)
1	0	1	0	1	0		10		250	0.083167
2	1	0	1	0	0		20		600	0.199601
3	0	1	1	0	0		12		304	0.101131
4	0	0	1	0	1		5		150	0.0499
5	1	1	0	1	0		26		906	0.301397
6	1	1	0	0	0		24		796	0.264804
								Average	501	

Table 4: Original members replaced by offsprings

The point to be noted here is that the average fitness, represented by f(x) here has improved from 383.33 to 501. Also, note that if the best individuals had been selected, the last bit would have a 1, which is not the optimal solution for our problem. Hence, the solution obtained would get stuck at a local minimum. To avoid this fitness scores are used and low fitness individuals are retained to maintain diversity.

Application in Data Mining

One of the major aspects of Data Mining includes the selection of variables, which is often critical in determining the performance of the model. The more popular algorithms such Gradient Boosting, GLMNET and Random Forest have automatic feature selection capabilities, but in cases when other algorithms where feature selection is not an automatic procedure, we need methods such as Genetic Algorithms for us to select the best features.

- Genetic Algorithm can be used as a wrapper around algorithms such as SVM and Ordinary Least Squares to select features, thus improving its performance in some cases or at least reducing the number of useful features.

- GA can be utilized to tune the weights assigned to the feature set while defining a KNN classifier. The weighting vector, containing weight for various features is modeled by encoding each feature as a distinct chromosome in the GA's schema. This weight vector can then be evaluated based on the error rates.

Case Study for comparison of Genetic Algorithms and Simulated Annealing

To learn about the practical application of Genetic Algorithms, the traveling salesman problem has been studied here.

Problem Statement : The problem at hand is that a salesman has to travel multiple cities in order to conduct his business. We need to find the sequence of travel in order to minimize it.

The type of problem is NP, which means that as the size of problem (number of cities) increase by n, the time and complexity increases by 2^n or n!

Given : Group of cities and distances between them

To find : The shortest path connecting each city once and returning back to the original city.

Approach 1 - Genetic Algorithm : We would use permutation encoding to encode each observation as string containing a travel pathway : For ex – ABEDCA . Multiple such combinations are created and the distances for each are calculated. The distances in this case is the fitness of each chromosome. After that, the well performing solutions are taken and either crossed over to produce offsprings, mutated to introduce change or both in sequence to obtain new candidate solutions.

So for example – IF CBCEDC and ADBECA performed, we could form offsprings at the second string value and mutate them. The offsprings would look like – CBAEDA and ADBECA. Their performance is evaluated again and the same steps are followed for multiple generations, where the good solutions are retained and the bad ones are discarded. In the figure below, we see that the solution moves towards optimality after a few hundred generations and ultimately flattens out.

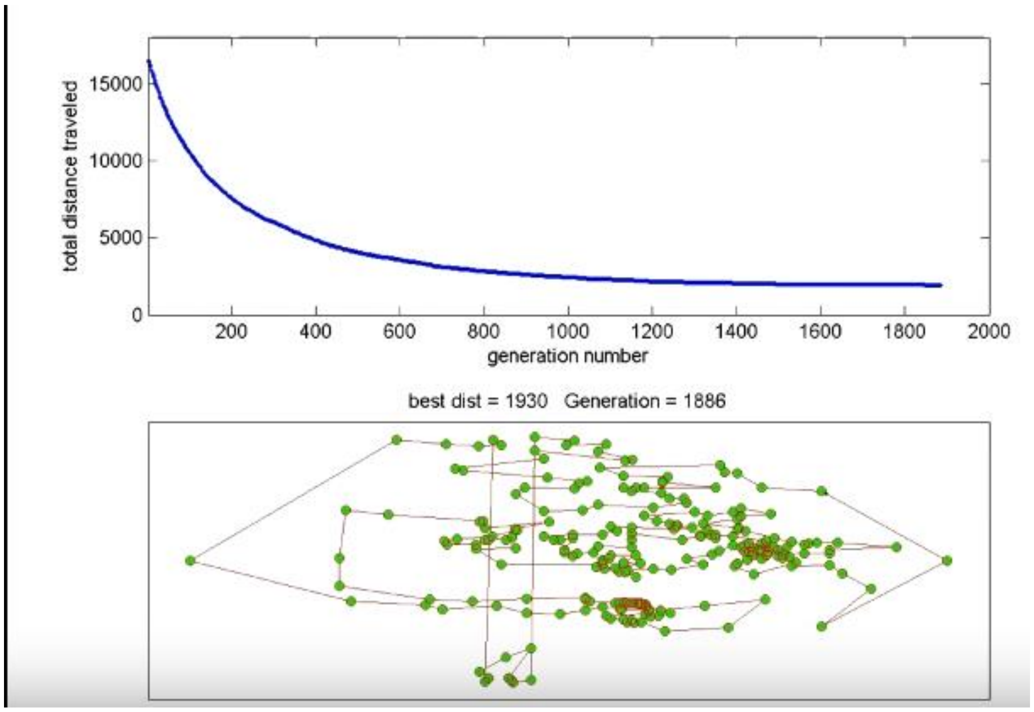


Fig3 :Reduction in distance travelled over generations,each green dot represents a city

Approach 2 - Simulated Annealing:

Background: Simulated Annealing comes from the concept of cooling a hot metal gradually, due to which the ions in the metal form a nice lattice structure.

For the traveling salesman problem, we first define a temperature T

Then, we form a path say ABCDEA and look at its adjacent path, which can be found by interchanging two letters in the path, say – ACBDEA.

The comparison is done in terms of the distance L_0 and L_1 for each path.

If $L_1 < L_o$, we will replace the adjacent path with the current path.

But if the $L_o < L_1$, we will replace our current path with the probability $e^{(Lo - L1)/T}$

The idea behind this is that if we directly select only the better path (lower distance in this case), we might converge to a local extremum (minima). Hence, we define T to jump over this local maxima points to obtain global extremum (minimum). To ensure that our solution converges to the local minimum and does not return back to the worst path, we reduce this temperature T at each time step.

Comparison against other techniques : Genetic Algorithms try to maintain this balance while other optimization algorithms lean one way or the other. Pure random search depends heavily on exploration of new and unknown areas. Hill Climbing is good at exploiting the information that it has, but not so effective in exploring unknown territories.

Genetic Algorithm implementation in R

For implementation, the Rastrigin function has been used since it is known that its global minima lies at (0,0). It is a common function used to test the optimization performance of algorithms. The **Rastrigin function** is a non-convex function used as a performance test problem for optimization algorithms. It is a typical example of non-linear multimodal function.

```r
Rastrigin <- function(x1, x2)
{
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}

x1 <- x2 <- seq(-5, 5, by = 0.1)
f <- outer(x1, x2, Rastrigin)
persp3D(x1, x2, f, theta = 50, phi = 20, color.palette = bl2gr.colors)
```

Fig 4: Coding the Rastrigin Function

We define the function and the bounds of x1 and x2 between (-5,5)

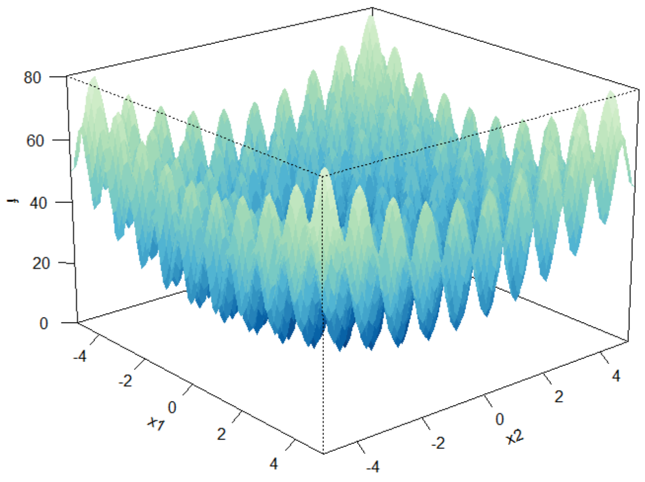
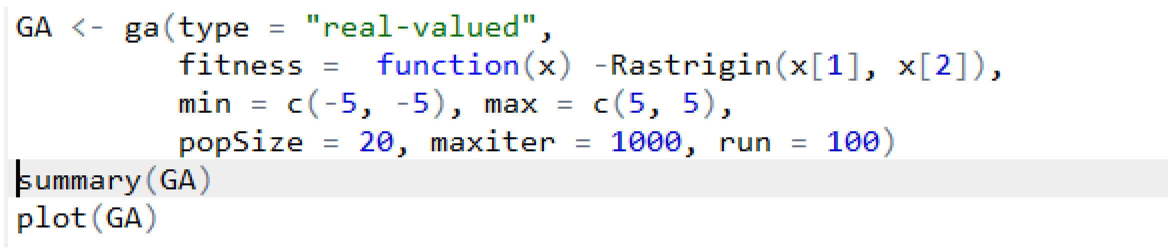


Fig 5: Rastrigin Function

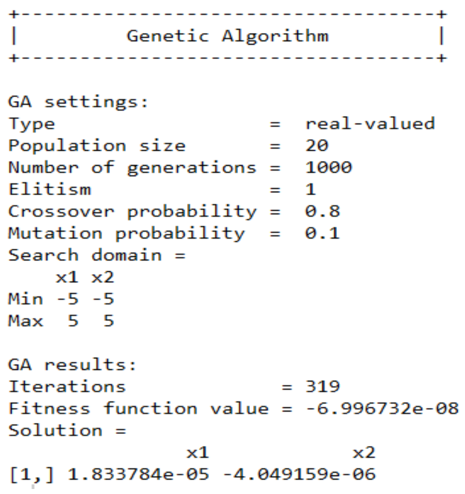
This is what the function looks like.



```
GA <- ga(type = "real-valued",
         fitness =  function(x) -Rastrigin(x[1], x[2]),
         min = c(-5, -5), max = c(5, 5),
         popSize = 20, maxiter = 1000, run = 100)
summary(GA)
plot(GA)
```

Fig 6: Code for defining GA parameters

We then define the GA function using the GA package in R. The lower and upper bounds, population size, the number of iterations and runs have been defined here.



```
+-----------------------------------+
|          Genetic Algorithm        |
+-----------------------------------+

GA settings:
Type                  =  real-valued
Population size       =  20
Number of generations =  1000
Elitism               =  1
Crossover probability =  0.8
Mutation probability  =  0.1
Search domain =
    x1 x2
Min -5 -5
Max  5  5

GA results:
Iterations             = 319
Fitness function value = -6.996732e-08
Solution =
               x1            x2
[1,] 1.833784e-05 -4.049159e-06
```

Fig 7: GA output

The solutions for x1 and x2 can be seen above. We see that they are very close to (0,0)

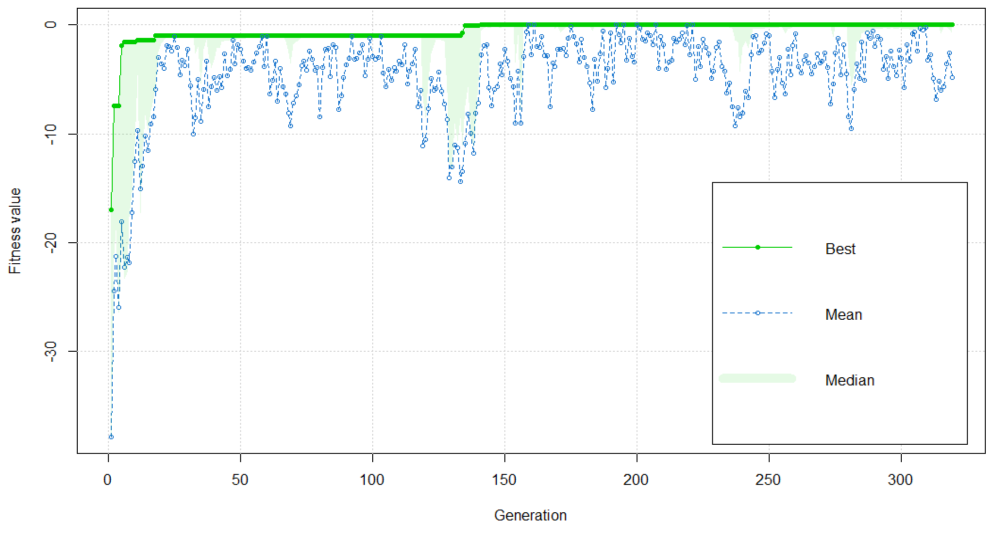


Fig 8: GA output over generations

Here we can see the performance of the Genetic Algorithm over several generations. It keeps moving towards the global minima over the generations.

Conclusion

To conclude it could be said that an efficient optimization algorithm is one that uses two techniques: exploration to investigate new and unknown areas in a search space and exploitation to make use of knowledge acquired by exploration to reach better positions on the search space, and Genetic Algorithm seems to find a balance between the two. However, one should be mindful of a few of the pitfalls of Genetic Algorithms- they are computationally expensive, sometimes undirected and are sensitive to initial parameters such as mutation rate.

But they are still a good method of obtaining an approximately optimum result and are more feasible compared to random search as the magnitude of the problem increases.

Further Study

Further study in this domain would include topics such as Hybrid GA's , which deal with incorporating efficient local search algorithms into GA's. Another potential topic of study could be 'Island evolution approach'- the population is partitioned in a set of subpopulations (islands) in which isolated GAs are executed on separated processor runs within a programming language. Occasionally, some individuals from an island migrate to another island, thus allowing subpopulations to share genetic material among each other.

References

http://geneticalgorithms.ai-depot.com/Tutorial/Overview.html

https://in.mathworks.com/discovery/genetic-algorithm.html

https://www.tutorialspoint.com/genetic_algorithms/index.htm

https://www.r-bloggers.com/genetic-algorithms-a-simple-r-example/

http://www.cs.utexas.edu/~dana/MLClass/GANotes.pdf

http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php

http://www0.cs.ucl.ac.uk/staff/C.Clack/PPSN98.pdf

https://www.youtube.com/watch?v=0rPZSyTgo-w&t=9s

R implementation: https://cran.r-project.org/web/packages/GA/vignettes/GA.html

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