Robotic Controller Design Using Genetic Algorithm

INFO 6205 - PROGRAM STRUCTURE AND ALGORITHM

FINAL PROJECT

NORTHEASTERN UNIVERSITY, BOSTON.

Author Name:

1) Dhanashri A. Palodkar - 001357556
2) Yogesh Suresh - 001237651

Professor:

Robin Hillyard

Abstract

Genetic Algorithm is an approach of solving constrained and unconstrained problems which are based on Natural selection using the Darwin's theory of evolution. The concept is derived from the genetic evolution and natural selection of human evolution and hence we can notice the overlap in technology. This algorithm repeatedly modifies the population of individual solution. At each step GA finds individual at random from population and consider it as parents. For further evolution. For example every solution has a value which we suppose to be gene as termed in biology every gene is attached with the other gene forming a chromosome and then determines the fitness of every chromosome. The chromosome which has the highest fitness score gives the information of its gene which is perfect fit for next generation. Over successive generation solution approach towards optimum solution.

Genetic Algorithm is one of the widely used algorithm since it works on varied field which includes the problem whose objective function is discontinuous, nondifferentiable, stochastic, or highly Non Linear.

Robot Controller Design Using Genetic Algorithm

Introduction

Problem Statement

Genetic Algorithm has wide use in robotics and AI to perform complex tasks with one algorithm removing need of manually coding for every function. In this problem we will apply genetic algorithm to design robot controller to perform actions (Movement) based on the inputs (Sensor), so the robot can navigate itself without need of directing it in an unknown environment. For example if we have a robot with N sensors to detect anomalies in physical space we need program the robot to take decisions based on 2^N action sequence base is 2 here since output of the sensor is 0 or 1. It is tedious to manually code the response for 2^N action sequence. So our objective is to use genetic algorithm to design robotic controller to respond to any given input action. In this project we will use maze structure as a testing environment for the designed robot.

In our problem, we are dealing with a robot controller which uses its sensors to navigate its way from the maze and avoid the obstacles. Depending on the Robots decision the robot will initiate the four actions which are it either turns left, turns right, go straight or does nothing with the help of sensors. Our task is to design the robot controller so that it can navgate itself.

.

Why we chose tournament selection?

Tournament selection selects its parents by running series of tournaments and also allows variable selection pressure. in our particular problem we are using sensor data which is variable, the tournament selection makes it easy to find the individual for the crossover since it selects individual based on its fitness value. This means that if the fitness of individual is higher the probability that the individual will be chosen for the crossover will be higher.

Why we chose Single point Crossover?

In Single Point Crossover , Randomly one single position is selected from the genome to define its genetic information which allows genetic information to be transferred from parents more effectively. Since In our problem the information of chromosomes is retrieved from the encoded input of 6 sensors and each instruction is more than one bit, single point processing was the best suits as it effectively transfer group of bits.

Terms used :-

Individuals : An Individual in genetic algorithm is an entity which can be compared to genes on which the fitness function is applied it is also referred as genome In our problem the individual or genome is action performed by robot. Also the value of fitness function for an individual is its fitness score.

Population - Population is an array of individuals, for example in our problem population is 1000 and the variable in fitness function is maze of 5X5 matrix then we have to represent population by 1000 in 5X5 matrix. Population is the collection where all genetic operators such as mutatio is crossover are applied.

Generation:- Genetic algorithm applies all the operation on population leading the new population after each iteration. Every successive population is called generation.

Chromosome – A possible solution to a given problem on which fitness function is applied to find optimum solution.

Gene – Gene is an element position of chromosome which can also be referred as parent.

Genotype − Genotype is the population present in computation space which can easily be manipulated. In our problem Genotype expression will be sensors data which is in binary value.

Phenotype − Phenotype is the population in the real world solution space in which solutions can not be manipulated. In this problem phenotype is the action performed by robot after getting the command.

Fitness Function − A fitness function is function which checks the suitability of the candidate solution as output.

Genetic Operators − These operators manipulate the genetic composition of population or offspring. For example, crossover, mutation, selection, etc.

Mutation – The process in which genes in a candidate solution are randomly altered to maintain the diversity in two successive generation

Crossover – The process in which chromosomes are combined to create a new candidate solution. This is sometimes referred to as recombination. There are various type of crossover e.g. Single point crossover, double point crossover. In our problem we are using single point crossover.

Selection – This is the technique of picking candidate solutions to breed the next generation of solutions.

Fitness Score– A score which measures the extent to which a candidate solution is adapted to suit a given problem.

Parameters:

Mutation Rate: It is the probability in which a parent gene of a chromosome will be mutated. If mutation rate is too low, the algorithm will take longer time to find the same solution.

Population Size:- It is the number of individuals present in one generation which 1000 in pur problem.

Crossover rate:- It is the frequency of crossover operation on one population. Higher the frequency the rate of finding new superior genes is higher.

Sample Robot Specification :-

Sensors Placement :

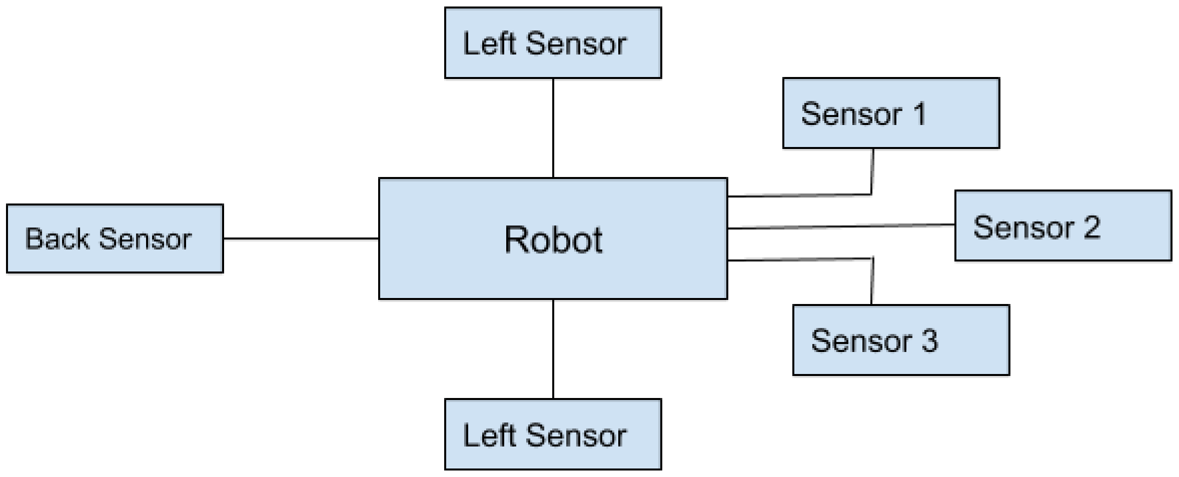


Fig 1.

Route Specification:

Let's take a 2D Environment of 5X5 which has following specification:

0 = Empty
1 = Obstacle/wall
2 = Starting position
3 = Route
4 = Goal position

The maze will be following

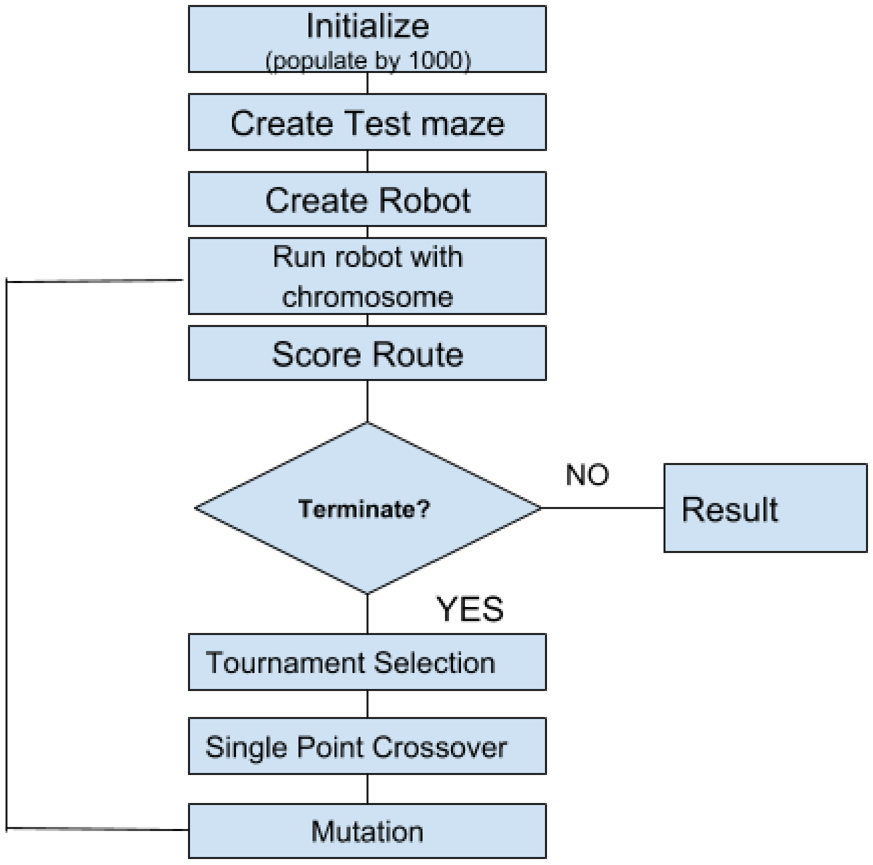
{1,0,1,0,2}
{1,1,1,1,3}
{1,1,3,3,3}
{3,3,3,1,0}
{4,1,1,1,1}

1	0	1	0	2
1	1	1	1	3
1	1	3	3	3
3	3	3	1	0
4	1	1	1	1

Design Flow

Following is the Program flow we have used in our project. In this project, We have taken a sample maze to solve where the robot will traverse the path from one point to other as aforementioned. Where the sensor data will give us the number of genes , we have 6 sensors wich give values in binary form on (1) and off (0) giving 2^6 = 64 combination. Since each combination encoding take 2 bits memory , the storage for this problem will be 64 * 2 = 128 bits. In genetic Algorithm chromosomes are easiest to manipulate as an array we will consider 128 as no. of chromosomes,

Stepwise Design flow is explained following



Design Flow

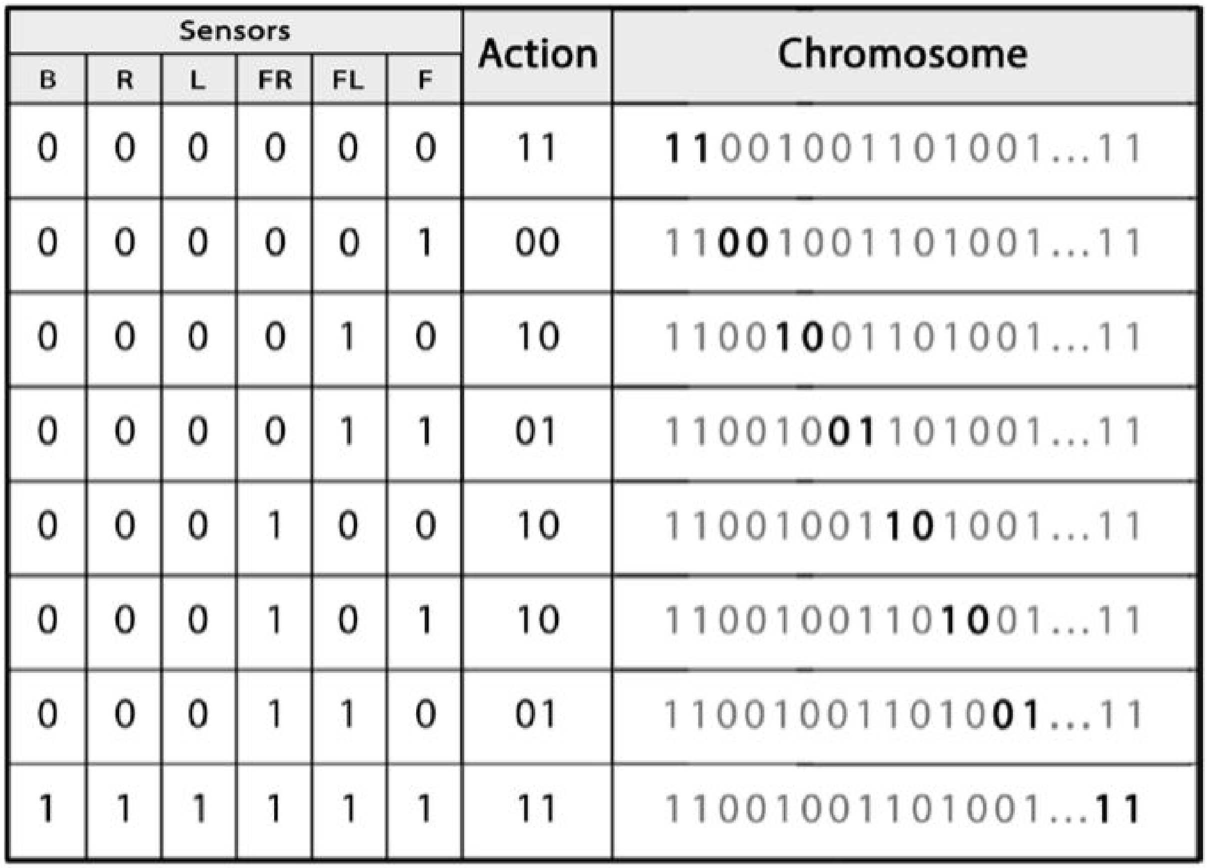
Fig 2.

Explanation

1. Initialize population : - in this problem, the genetic Algorithm starts with the population of 1000 generations which helps us to create the maze.
2. Create Maze: - The test maze is created using the population given where 3 will show the path that robot has covered to reach from starting point 2 to goal point 4.
3. Run robot :- Run function is called with 128 chromosomes which it gets from the sensor inputs.
4. Score Route :- It will evaluate the route and compare with best route for fitness.
5. Termination check :- If the Generation count is less than the maximum generation it will go to tournament selection else it will give the final result
6. Tournament Selection :- Tournament selection provides a method for selecting individuals based on their fitness value. That is, the higher an individual's fitness the greater the chance that the individual will be chosen for crossover
7. Single Point Crossover :- Single point crossover is a very simple crossover method in which a single position in the genome is chosen at random to define which genes come from which parent.
8. Mutation : - Mutation is used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.

Relation Between GenoType and Phenotype :

The expression between Genotype and Phenotype is defined as follows:

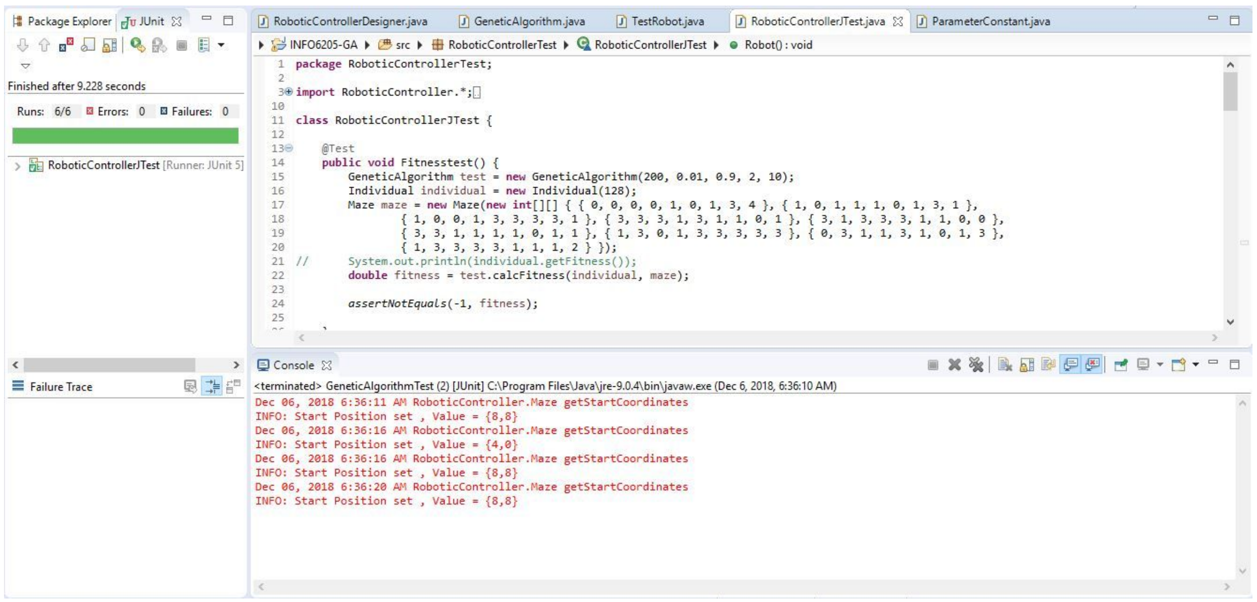


Sensors						Action	Chromosome
B	R	L	FR	FL	F		
0	0	0	0	0	0	11	**11**001001101001...11
0	0	0	0	0	1	00	11**00**1001101001...11
0	0	0	0	1	0	10	1100**10**01101001...11
0	0	0	0	1	1	01	110010**01**101001...11
0	0	0	1	0	0	10	11001001**10**1001...11
0	0	0	1	0	1	10	1100100110**10**01...11
0	0	0	1	1	0	01	110010011010**01**...11
1	1	1	1	1	1	11	11001001101001...**11**

Fig 3.

Results (Unit Test Case)

To check the Working condition of code implemented for genetic Algorithm we had written 6 Test Cases (refer :https://github.com/yogeshsuresh93/INFO6205_230). The Test cases were tested against each function used in genetic Algorithm. The test case result is as follows:


Finished after 9.228 seconds

Runs: 6/6 Errors: 0 Failures: 0

RoboticControllerJTest [Runner: JUnit 5]

```java
package RoboticControllerTest;

import RoboticController.*;

class RoboticControllerJTest {

    @Test
    public void Fitnesstest() {
        GeneticAlgorithm test = new GeneticAlgorithm(200, 0.01, 0.9, 2, 10);
        Individual individual = new Individual(128);
        Maze maze = new Maze(new int[][] { { 0, 0, 0, 0, 1, 0, 1, 3, 4 }, { 1, 0, 1, 1, 1, 0, 1, 3, 1 },
                { 1, 0, 0, 1, 3, 3, 3, 3, 1 }, { 3, 3, 3, 1, 3, 1, 1, 0, 1 }, { 3, 1, 3, 3, 3, 1, 1, 0, 0 },
                { 3, 3, 1, 1, 1, 1, 0, 1, 1 }, { 1, 3, 0, 1, 3, 3, 3, 3, 3 }, { 0, 3, 1, 1, 3, 1, 0, 1, 3 },
                { 1, 3, 3, 3, 3, 1, 1, 1, 2 } });
//      System.out.println(individual.getFitness());
        double fitness = test.calcFitness(individual, maze);

        assertNotEquals(-1, fitness);

```

```
<terminated> GeneticAlgorithmTest (2) [JUnit] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (Dec 6, 2018, 6:36:10 AM)
Dec 06, 2018 6:36:11 AM RoboticController.Maze getStartCoordinates
INFO: Start Position set , Value = {8,8}
Dec 06, 2018 6:36:16 AM RoboticController.Maze getStartCoordinates
INFO: Start Position set , Value = {4,0}
Dec 06, 2018 6:36:16 AM RoboticController.Maze getStartCoordinates
INFO: Start Position set , Value = {8,8}
Dec 06, 2018 6:36:20 AM RoboticController.Maze getStartCoordinates
INFO: Start Position set , Value = {8,8}
```

Test Case 1 (Fitnesstest()) :-

Use Case: To Test Custom Fitness Function in GA

Condition given : calcFitness is run hence the default fitness present in GA will be updated

Expected Result: fitness value must not be equal to default value

Test Case 2 (Mutationtest()) :-

Use Case: To Test Custom Mutation Function in GA

Condition given : mutateFuction is run hence the new population is generated

Expected Result: Newly generated population must not be same as old population

Test Case 3 (Crossovertest()) :-

Use Case: To Test Custom Crossover Function in GA

Condition given : crossoverPopulation is run hence the new population is generated

Expected Result: Newly generated population must not be same as old population

Test Case 4 (Fitnesstest2()) :-

Use Case: To Test Custom Fitness Function in GA

Condition given : GA is run with required parameter hence the end fitness value is best score route of the maze.(Note this is not to infer best path taken but optimum chromosome is created)

Expected Result: fittest individual in the population will have best score route of given maze.

*best score means the route taken by the robot is what we have designed in the input maze.Recall 3 value in maze array defines the best tiles to step in by robot.

Test Case 5 (Robot()) :-

Use Case: To Test Run() in Robot in GA

Condition given : Robot is run using best chromosome generated by GA in new maze i.e the designed controller is now put into new test environment to check if robot is not hitting any obstacle

Expected Result: Robot route final path must not be an obstacle/wall so it is infered that robot has avoid the obstacle in its course of travel.

Test Case 6 (scoreRouteTest()) :-

Use Case: To Test Custom scoreRoute Function in GA

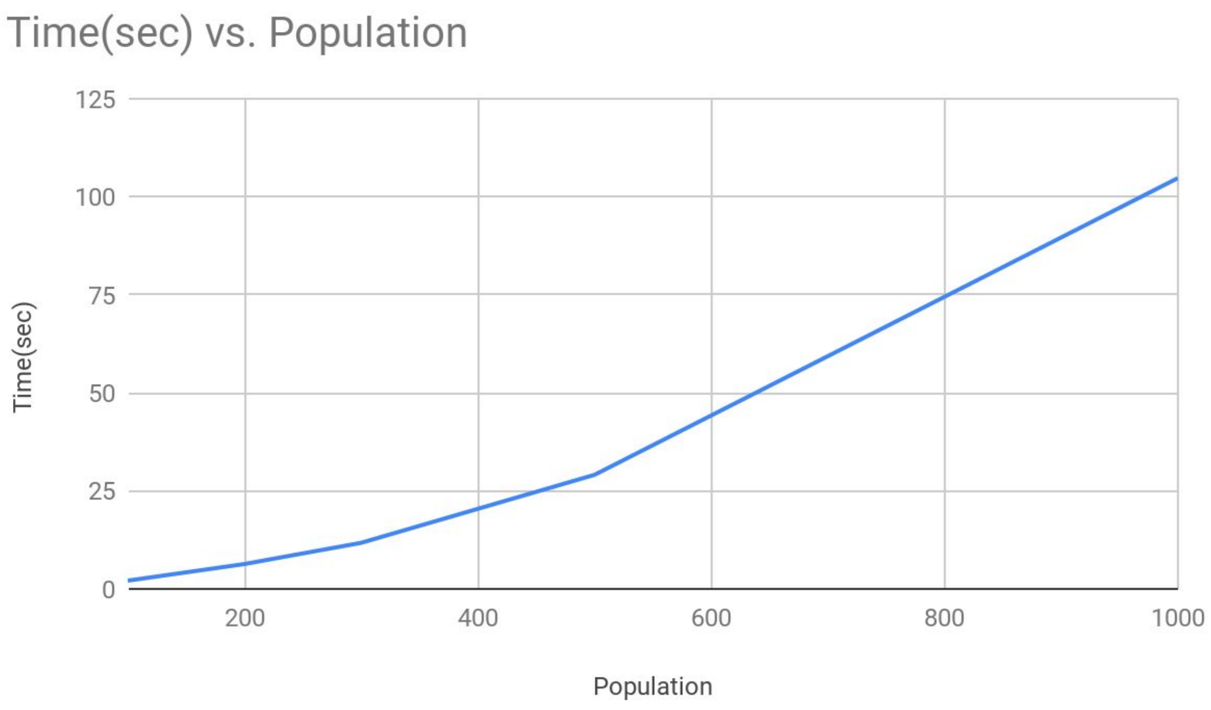
Condition given : Best route is given to the function so it must return no of best tiles

Expected Result: Returned score must be equal to no of best tiles that we set in the maze

Outcome 1

Following is the outcome tested for time vs population Size:

Population	Time(sec)
100	2.105
200	6.344
300	11.747
500	29.08
1000	104.741

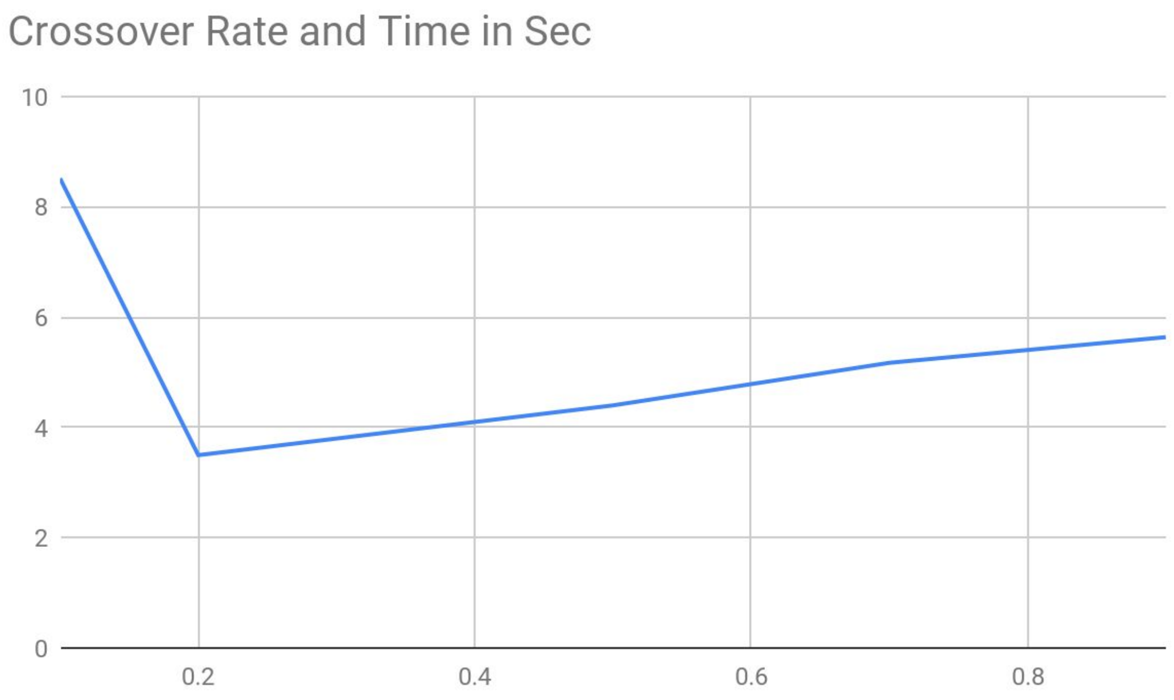


Observation :
It is observed that as the no. of population increased the time of execution is also increased Although,the best fit solution was found earlier generations itself when implemented with the bigger population

Outcome 2

Outcome 2 shows graph for Crossover rate vs Time

Crossover Rate	Time in Sec
0.9	5.636
0.7	5.171
0.5	4.399
0.2	3.497
0.1	8.514

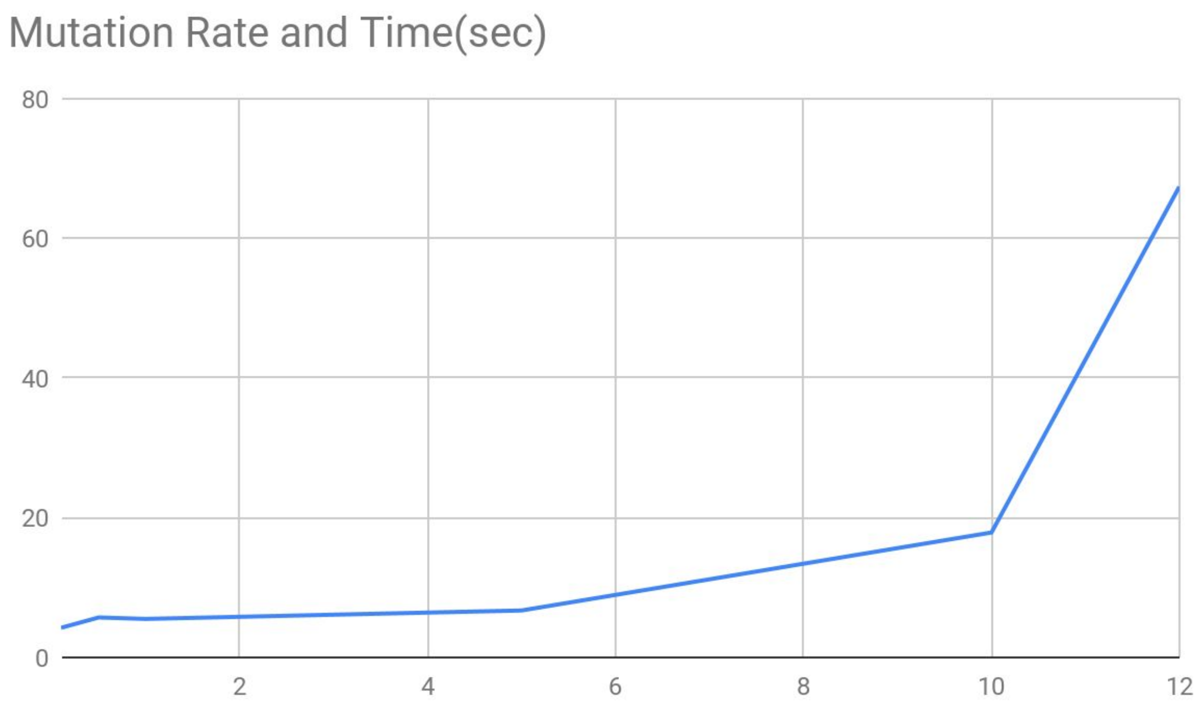


Observation :
It is observed that for very low crossover rate GA has high execution time but after certain rate is almost linear. This is what the is inferred from the GA with low crossover rate the diversity is too low so evolution is at very less rate.

Outcome 3:

It shows Mutation rate vs Time in Sec.

Mutation Rate	Time(sec)
0.1	4.246
0.5	5.722
1	5.514
5	6.721
10	17.883
12	67.399



Observation :
It is observed that as the Mutation rate increased the time required to run the program increased. Mutation is used to maintain the diversity in every generation and it is observed that mutation is important to converge th GA.

Conclusion

We have learnt the practical application of Genetic Algorithm which is helpful to arrive a solution with N-dimension problem space. Here we had 2-dimension use case with four resultant sequence, mapping for this particular problem require 2^N * 2(2-bit encode for four action sequence) solution set. For larger value of N i.e dimension we find GA effective though it is not 100% best solution . With evolution theory we find converging solution in few mins/sec instead of million or billion more years if it is manually commuted and solved.

References

1] Jacobson, L., & Kanber, B. (2015). Implementation of a Basic Genetic Algorithm. In *Genetic Algorithms in Java Basics* (pp. 21-45). Apress, Berkeley, CA.Lastname, C. (2008). Title of the source without caps .fig [3]

2] https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.html

3] https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications