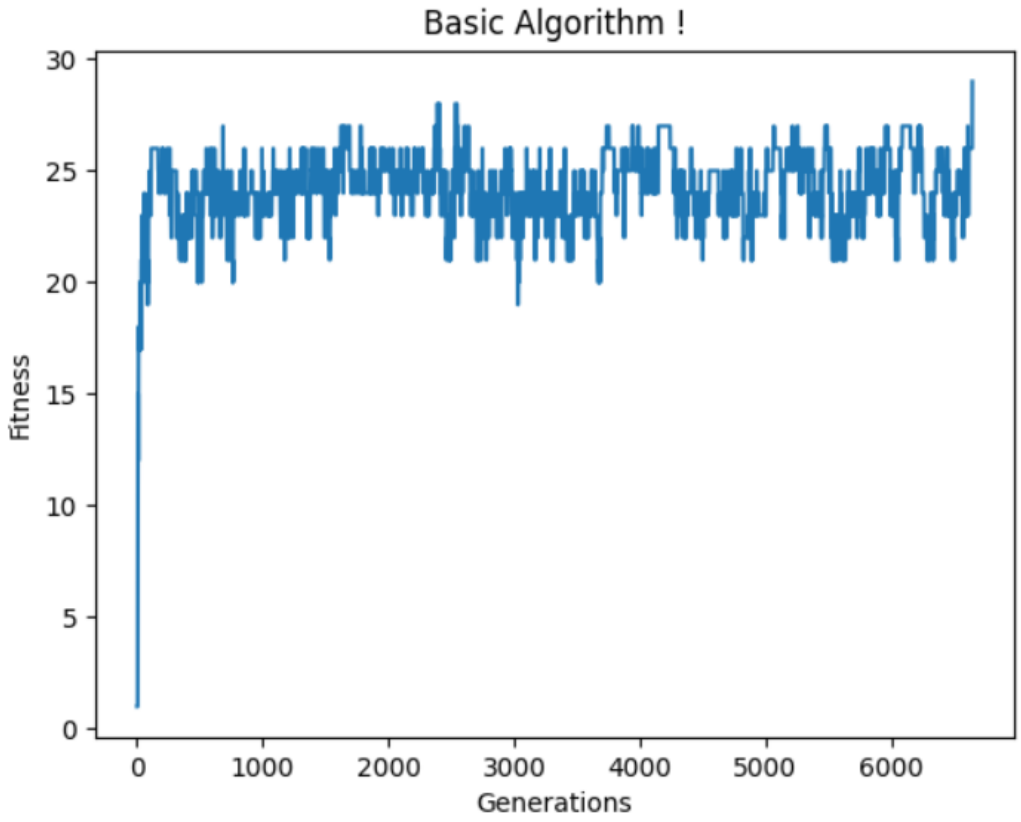
Report for Assignment 1

Name -> **Mayank Sheoran**
ID -> **2018A7PS0114G**

I have created 2 classes [1 for each problem] , user can choose which problem to solve by giving input 1 & 2 for 8 Queen And TSP respectively

Question 1:- 8-Queen Problem

First I have implemented the Basic Book algorithm which although does not give promising results and graph for the same is as follows.



BASIC ALGORITHM

Population Size = 20
Fitness Function -> Calculating the attacking positions on ROWS , COLS (0 As per our assumption that all queens will move only in their respective column) , Left diagonal , Right Diagonal. Then we can calculate fitness by subtracting Total Combination - Attacking Positions
Mutation function -> Just changing the random column with random value.
getParent function -> Selecting 2 Parents from population on the basis of their fitness.
getChild function -> Selecting random index and swapping parents across that index .
Achieving the BEST FITNESS in AVG of **7000 Generations**

[

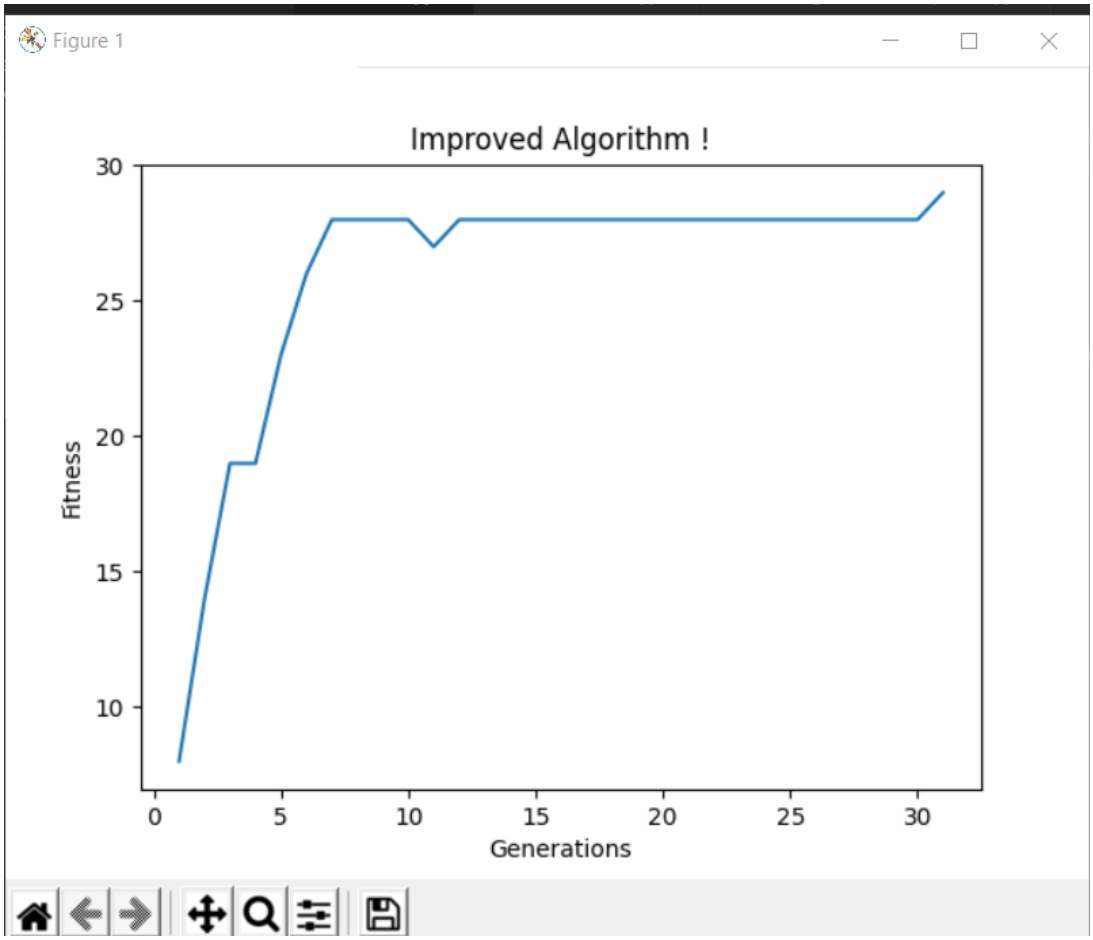
PROGRAM STOPS ONLY WHEN IT ACHIEVES THE BEST FITNESS OF 29
Because this algorithm is achieving best fitness in less generations so i have not restricted the algorithm to stop after a particular amount of generations.

]

IMPROVED ALGORITHM

I have tried multiple algorithms in this question but found that Normal hill climb is working best in this question.
(I have commented the part where i have tried the stochastic hill climb but results are not as promising in Normal Hill climb)

Population Size -> 40
Fitness Function -> Same
Mutation Function -> Used Hill Climb Algorithm to Find the best neighbour around our child which is being mutated and if we are at the position where we are the best among our neighbours then we mutate randomly which is same as basic Algorithm
How used -> Taking all possible indexes and all possible row values for that Index column. And selecting a child with best fitness.
getParent function -> Same As Basic Algorithm Choosing 2 parents according to their respective weights/probability.
getChild function -> Used Hill Climb and finding the best neighbour.
How Used -> Iterating on all possible indexes and then swapping the 2 parents across that index and choosing the best child among all neighbours and if this child is the same as any parent then we go with a basic approach which is random selection of indexes.

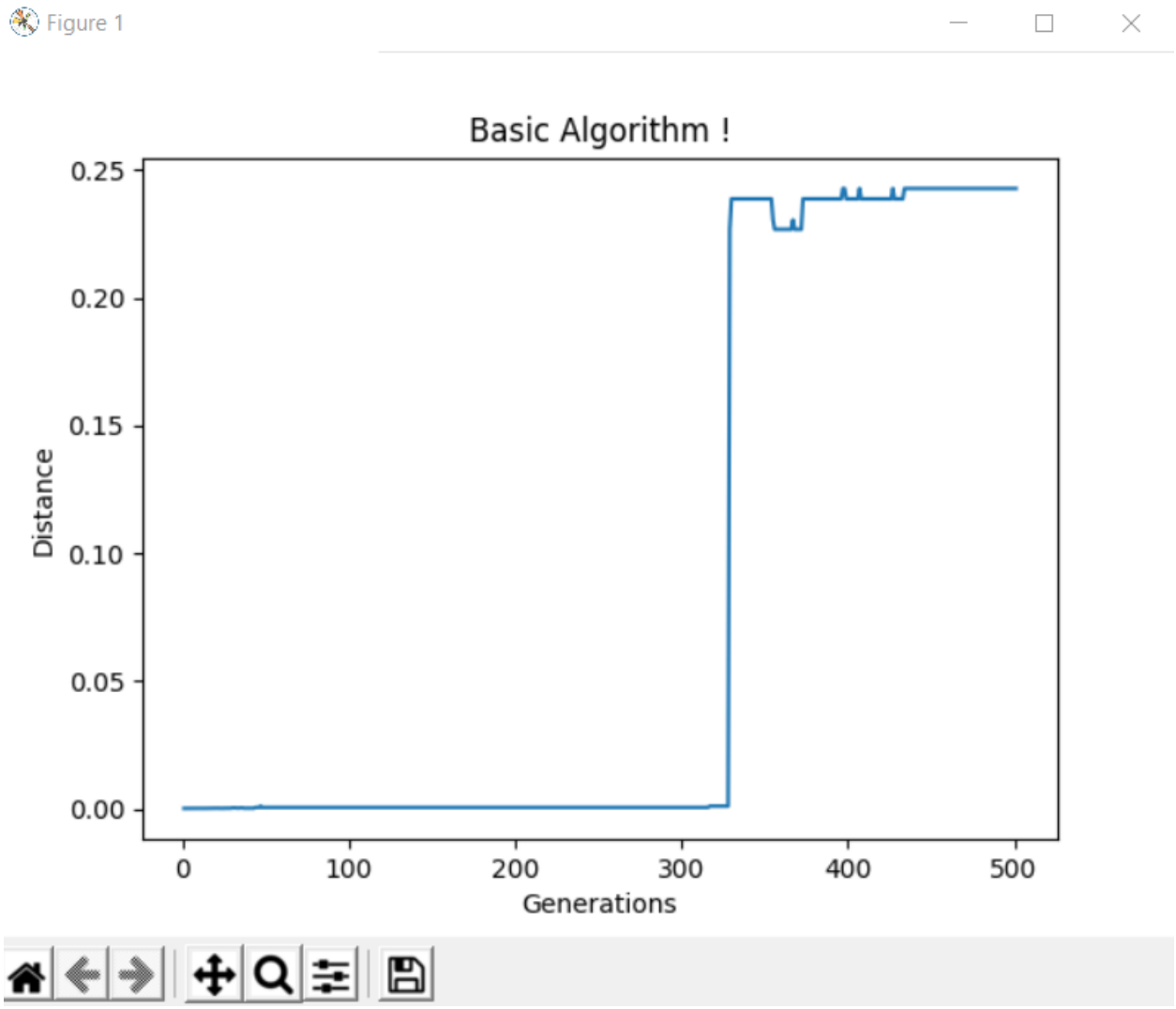


Achieving the BEST FITNESS in AVG of **30 Generations**

Question 2:- TSP Problem

First I have implemented the Basic Book algorithm which although does not give promising results and graph for the same is as follows.

In this problem we will compare the first 500 generation for both algorithms using graphs



BASIC ALGORITHM

Population Size = 20
Fitness Function -> We are calculating the distance for any given state , for example if state is [A,B,C …. N] then distance will be calculated as follows
Total Dist = Dist(A-B) + Dist(B-C) + ……. Dist(M-N) + Dist(N-A)
And 1/TotalDist is our Fitness.
Mutation function -> Choosing 2 indexes randomly and then swapping the cities at these indexes.
getParent function -> Selecting 2 Parents from population on the basis of their fitness.
getChild function -> Selecting random index to select the start point and random Length which signifies the length of parent1 part

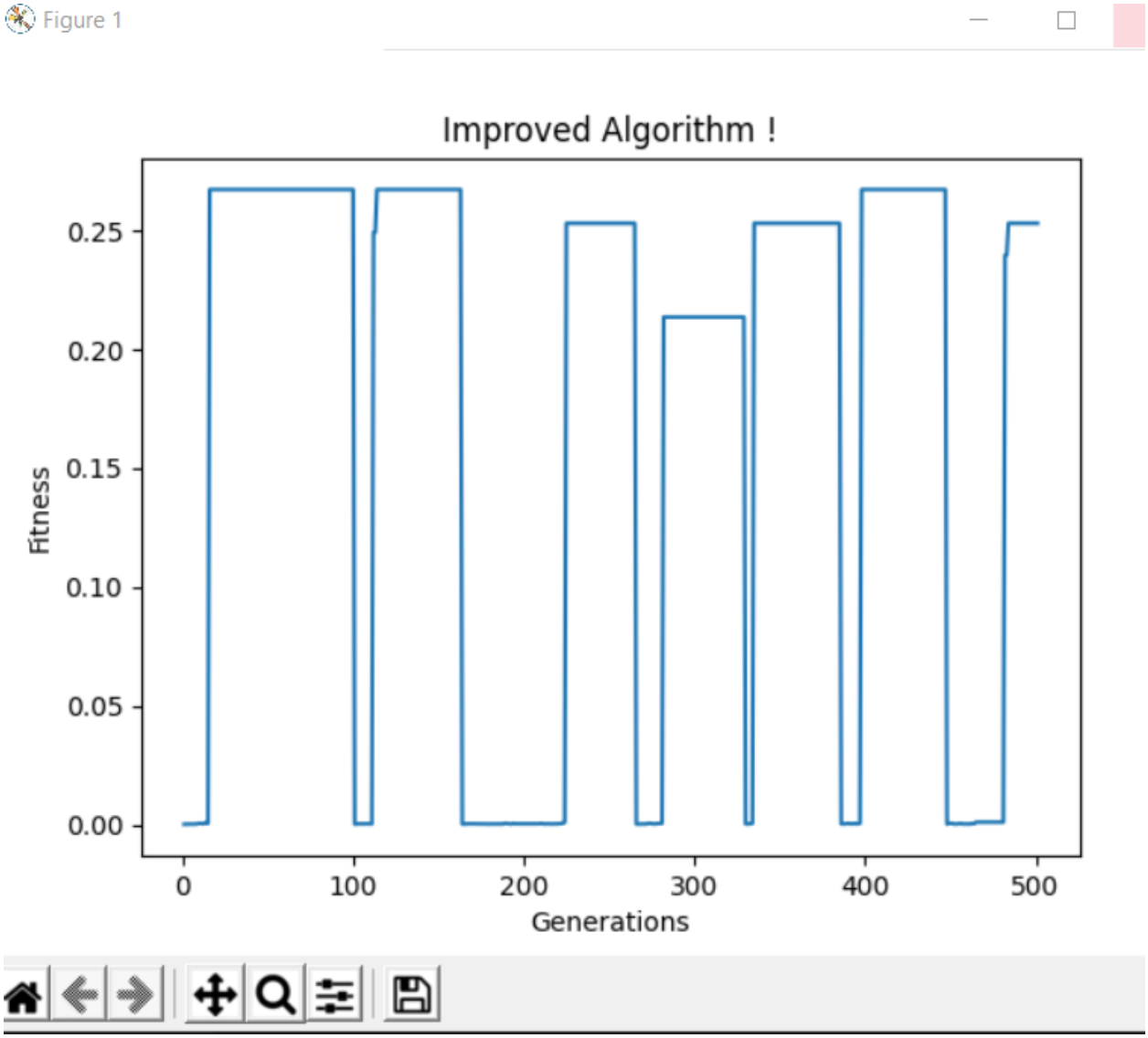
Now we are taking the parent1 Part from **Index -> index+length**, this part is given to child with the same indexes **index -> index + length** and the rest of the indexes of Child will be filled with parent2 cities which are not in common with parent1 Part.

Achieving the BEST FITNESS in AVG of **1000-1200 Generations**

IMPROVED ALGORITHM

Population Size -> 20
Fitness Function -> Same
Mutation Function -> Used Hill Climb Algorithm to Find the best neighbour around our child which is being mutated and if we are at the position where we are the best among our neighbours then we mutate randomly which is same as basic Algorithm
How used -> Selecting 2 indexes index1 , index2 using 2 for loops and then using these indexes we are making a tempCHILD (By swapping cities at these indexes) which is then compared with our original child and then selecting the best neighbour. And if we are already the best then we randomly mutate the child by random index.
getParent function -> Same As Basic Algorithm Choosing 2 parents according to their respective weights/probability.
getChild function -> Used Random Hill Climb Algorithm
How Used -> Choosing all possible indexes and all possible lengths for selecting the parent1 Part and then creating the tempCHILD for all these reproduces childs and then selecting the best and if we are the best again going with the basic method.

But in this method i have changed 2 things in genetic algorithm ,
First , I am increasing the probability to mutate by a small amount if we are getting the same fitness in our past 2 generations and this way there will be more probability to mutate but if the last 2 generations do not have the same fitness then we will reset the probability to mutate to default value.
Second , There are sometimes where i get stuck at particular fitness for long number generations even after multiple mutations so to tackle this i maintained the RepeatCOUNT which is maintaining the count that how many past generations are getting the same fitness this way i can solve the issue by randomly shuffling the population and again starting the shuffled populations
Benefit for this is that if we are not stuck at all then we will get our answer in very less number of generations like less than 60/70 but if we are stuck then we might get stuck for another 100 generations therefore by shuffling and restarting we are able to get out of this problem and after 1 or max 2 restarts we are always getting out best fitness/ Least distance.



If we compare the graph of basic and improved algorithms we can clearly see that we are hitting our target in less than 100 generations and if we are stuck then we restart and get our target in another try.

Achieving the BEST FITNESS in AVG of **50 Generations**