Author: Melwyn D Souza
Student Number: R00209495
Email: melwyn.dsouza@mycit.ie
Course: MSc Artificial Intelligence
Module: Metaheuristic Optimization
Date: 07/11/2021

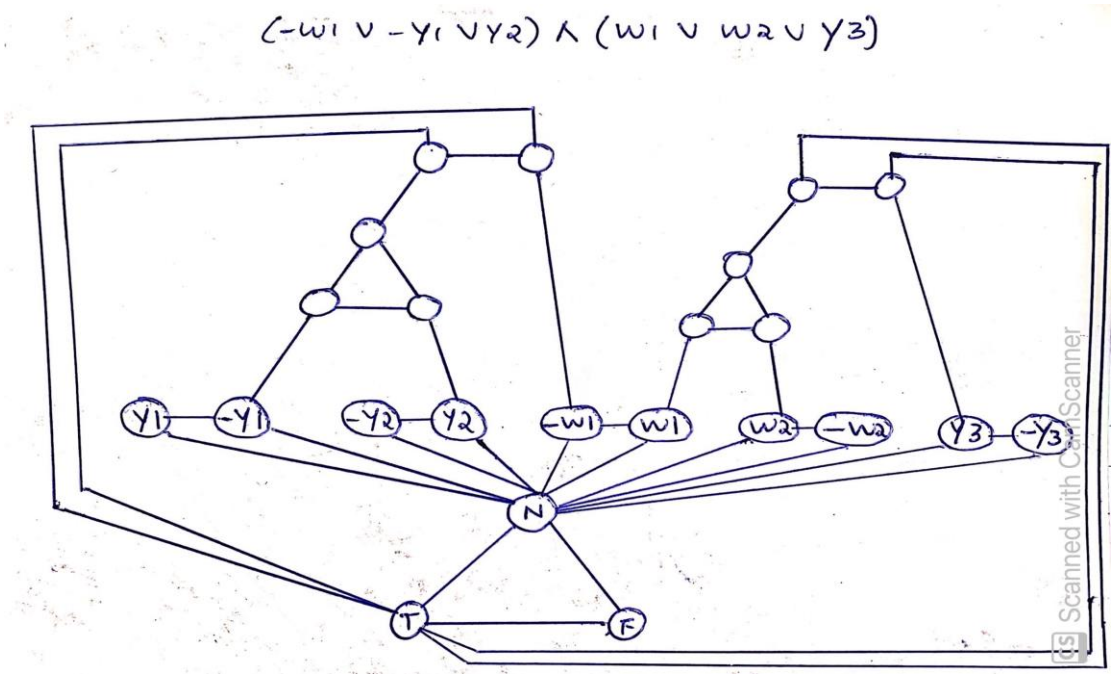
1.1 NP – Completeness

Q1. Last digit of student id is **5 (R00209495)**, therefore selected (b) F = (-w1) ^ (w1 V -w2 V w3 V -w4 V w5) from assignment1

Q2. 2nd and 5th clauses selected from F` since the first letter of my first name is in the range J-R **(Melwyn)**

1. This problem concerns the proof of the NP-completeness of 3-SAT
   a. Convert F = (-w1) ^ (w1 V -w2 V w3 V -w4 V w5) into 3-SAT formula using construction/reduction
      i. Clause1 = (-w1)
      ii. New clauses from Clause1 = (-w1 V -y1 V -y2) ^ (-w1 V -y1 V y2) ^ (-w1 V y1 V -y2) ^ (-w1 V y1 V y2)
      iii. Clause2 = (w1 V -w2 V w3 V -w4 V w5)
      iv. New clauses from Clause2 = (w1 V w2 V y3) ^ (-y3 V w3 V y4) ^ (-y4 V w4 V w5)
      v. F` = (-w1 V -y1 V -y2) ^ (-w1 V -y1 V y2) ^ (-w1 V y1 V -y2) ^ (-w1 V y1 V y2) ^ (w1 V w2 V y3) ^ (-y3 V w3 V y4) ^ (-y4 V w4 V w5)
   b. Find a solution for F` and verify it's a solution for F
      i. w1 = F, w2 = F, w3 = F, w4 = F, w5 = T, y1 = F, y2 = F, y3 = T, y4 = T
      ii. F` = (T V T V T) ^ (T V T V F) ^ (T V F V T) ^ (T V F V F) ^ (F V F V T) ^ (F V F V T) ^ (F V F V T)
      iii. Plugging the values found above to the original formula
      iv. F = (T) ^ (F V T V F V T V T)
      v. Every clause in F and F` has at least 1 True literal given the solution

2. Convert subclauses from F` to 3COL graph (2nd and 5th clauses)



1.2 Genetic Algorithm (GA)

For the GA algorithm development, debug, testing and capturing results, three instance files are used, namely, inst-0.tsp, inst-13.tsp, inst-5.tsp based on the first letter of my surname (D Souza)

Introduction

GA is a heuristic algorithm which helps in optimizing the problem solutions, it is a part of evolutionary algorithms, the mechanics of it are based on natural selection and genetics
The evolution is based on Darwin's survival of the fittest theory, where the stronger survive and pass on their traits to their offspring's/future generations and the weaker die

Travelling salesman problem (TSP)

Travelling salesman problem is a search problem where a salesman starts from one city and tries to travel to the closest next city, this way the salesman must cover every city only once and return to the city that he started from
The main agenda of TSP problems is to find the shortest path for the salesman to travel all cities and return to first city, we can use different heuristics to solve these problems, here we will discuss about a simple genetic search algorithm which is used to optimize the solutions that random initialization and nearest neighbour initialization heuristic produce

Methodology

- Step 1: Initialization - Create a population with N number of chromosomes and compute fitness, use random or nearest neighbour heuristics to find the chromosomes (TSP solutions) for our population

- Step 2: Sort the population based on the fitness (TSP shortest path tour is first and the list keeps ascending in fitness value with the worst solution at the end of the list), a pool is used called mating pool to get a copy of all the chromosomes present in the current generation and sort them for execution of next steps

- Step 3: Selection - Truncation selection method which is a rank-based selection is used to select the parents from the mating pool, the selection is carried on depending on the truncation percentage, for example trunkP – 0.1 means, select the parents from the top 10% of the best performing chromosomes

- Step 4: Crossover – Cross over is equal to mating two parents to produce a child for the next generation, in our case order 1 crossover is applied, a random segment is

selected from parent 1, the genes from parents 2 which are present in the segment are then deleted, the rest remaining genes are pushed to the left and the segment is then appended to the end to produce one child, the same can be repeated and child 2 can be produced, the crossover is applied depending on the crossover probability pC

- Step 5: Mutation – Mutation is altering few genes in the child inorder to mutate the child chromosome, this sometimes helps in overall reduction of the fitness, we can see this later in the experimental results and graphs, the mutation is applied depending on the mutation rate pM

- Step 5: Replacement – This is a technique which is used to retain some of the best performing parents in the current generation and pushing them to new generation without crossover or mutation or any other alterations, I have used elitism, depending on the elitism probability specified by the user, the elites or the top performers of the current generations will be copied over to the next generation without altering their genes

Experimental results

Impact of initialization method on performance

Two initialization techniques are used in initializing the population, random initialization, and nearest neighbour initialization

In the random initialization technique, random cities are selected one after the other and a chromosome is created, since this technique does not calculate the Euclidean distance before adding the next city to the tour, the total distance will be very large as seen in the initial best solutions column in the table below, the GA has good room to optimize the solution and get the best solution from a randomly initiated population, this can be seen in the final best solution for different instance files in the table below

Nearest neighbours technique starts with randomly selecting one city from the instances and calculates the Euclidean distance from the current city to all the cities, it then appends the next city at shortest distance from the current city, it continues this loop until a full chromosome is formed, as can be seen in the table below, the initial solutions for NN initialized populations is way better that randomly initialized population, the GA has very less room for improvement since the population and the chromosomes in the population are already very fit, that is the reason for very low performance of GA over NN compared to random initialization

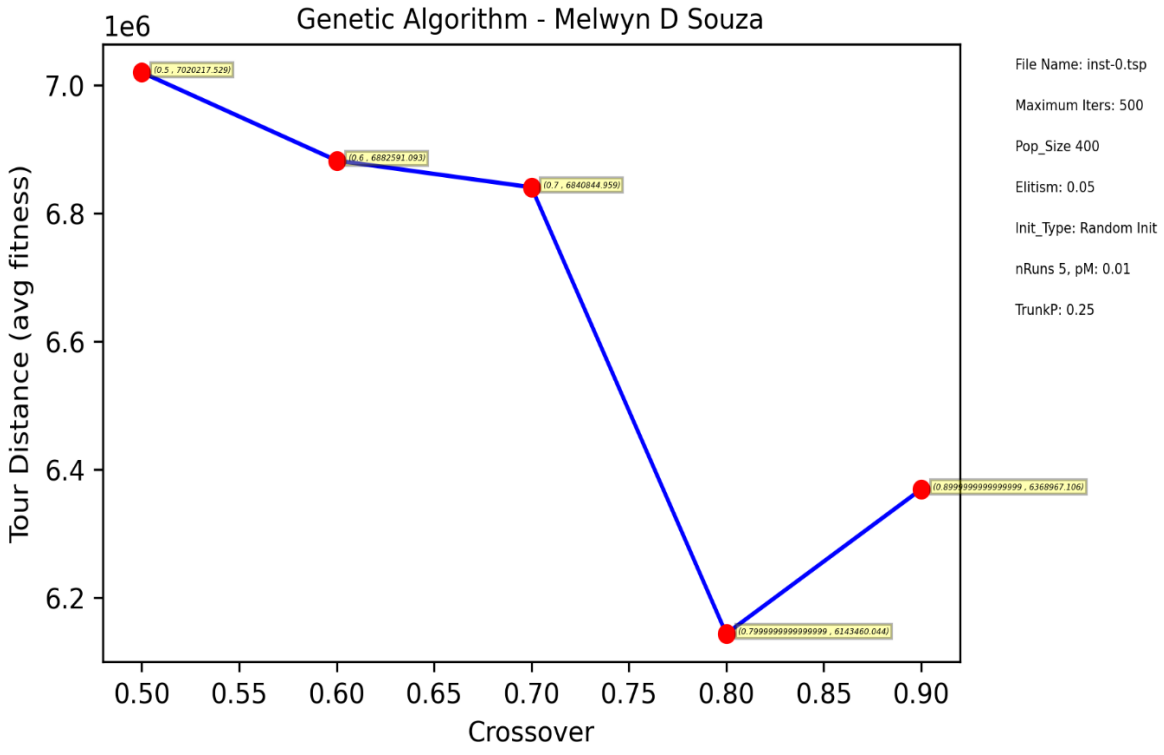
Initialization type	File name	Initial best solution (average)	Final best solution (average)	nRuns
Random	inst-5.tsp	431418204	107493169	10
	inst-13.tsp	113384792	21585461	10
	inst-0.tsp	22932318	7330982.6	10
NN	inst-5.tsp	12965792	12834404	5
	inst-13.tsp	7206705	7096483.6	5
	inst-0.tsp	3992002	3834585	5

Impact of crossover on performance

Crossover is applied on 2 parents selected from the mating pool, the probability pC decides how often crossover is to be applied, for example, if crossover probability is 0.8, 80% of the times, the parents are crossed over and children are produced and 20% of the times, one of the parents is considered a child without any crossover

The results that I have obtained from my experiment were conducted on instance file inst-0.tsp, this was to get a better understanding of the impact of the crossover on the TSP GA, it can be seen from the graphical representation below where Y axis represents the average fitness obtained over the specified nRuns and X axis represents the pC

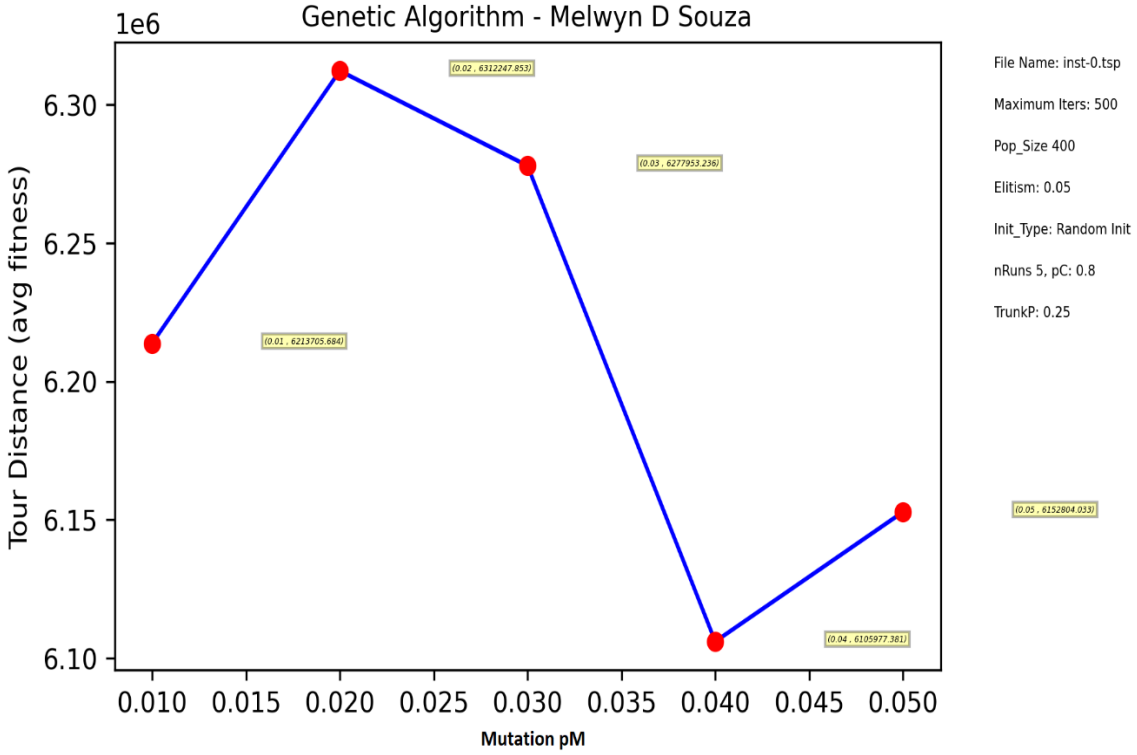
All other features are added to the top right corner of the graph as shown below, we can notice that as pC increases slowly, crossover helps GA find the best solutions, and from my experiment, I have selected 0.8 to be ideal pC for my other runs



Impact of mutation on performance

There are different types of mutation techniques in GA, I have selected Inversion mutation for my experiments, Inversion mutation selects a random range in the chromosome and reverses it producing a mutated child, the mutation is applied on the population by the mutation probability pM, as discussed above for crossover, mutation acts the same on percentage distribution over the population, for example pM – 0.01 means, only mutate 1% of the population

I have varied mutation probability (X axis) and calculated the best solutions obtained (Y axis) over many iterations as shown in the graph below, we can see that the value pM = 0.04 has served my GA with best results, but we need to keep in mind that if the mutation percentage is too large, we might lose the best or fittest chromosomes and the performance will drop drastically



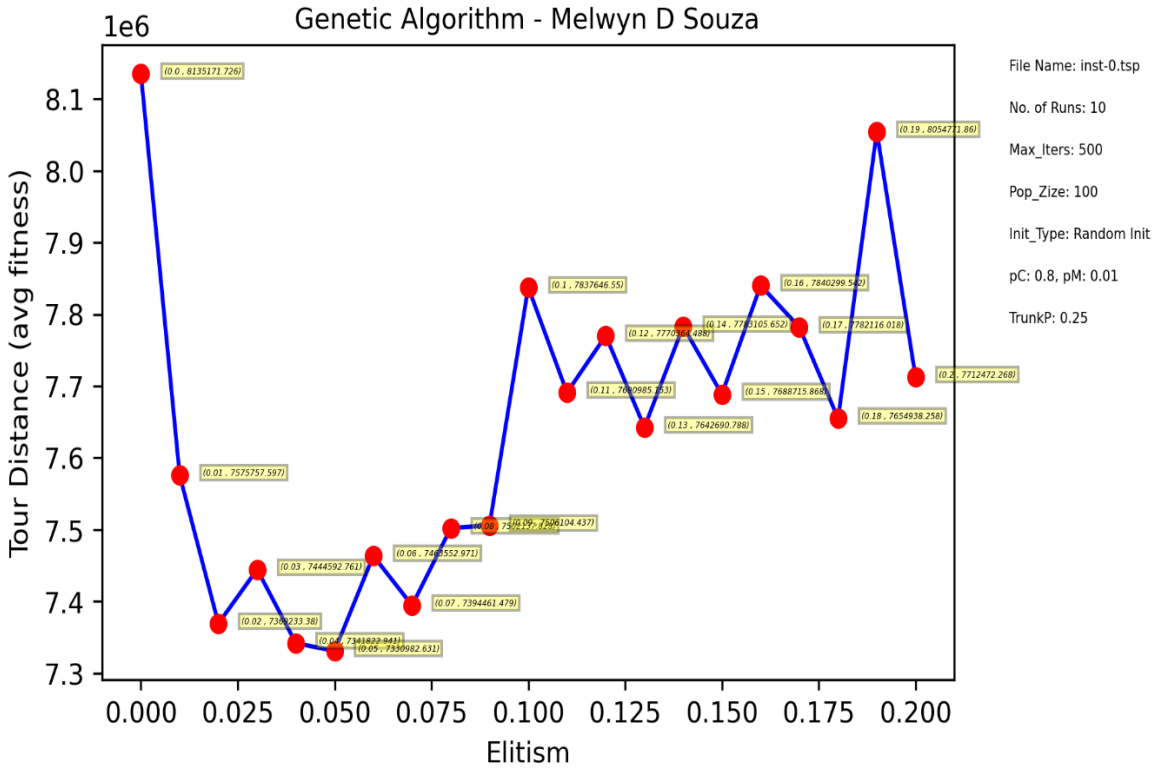
Impact of elitism on performance

Elitism is a replacement strategy, the elite probability is to be given by the used which defines the percentage of the top performers that will be selected from the current generation, these top performers will be copied over to the new generation without any crossover or mutation

It is good to use elitism since the parents selected for mating might be more fitter than the child produced, and if we don't retain these parents, they will be lost forever in the new generation, but we also need to be careful of the convergence of the population, if

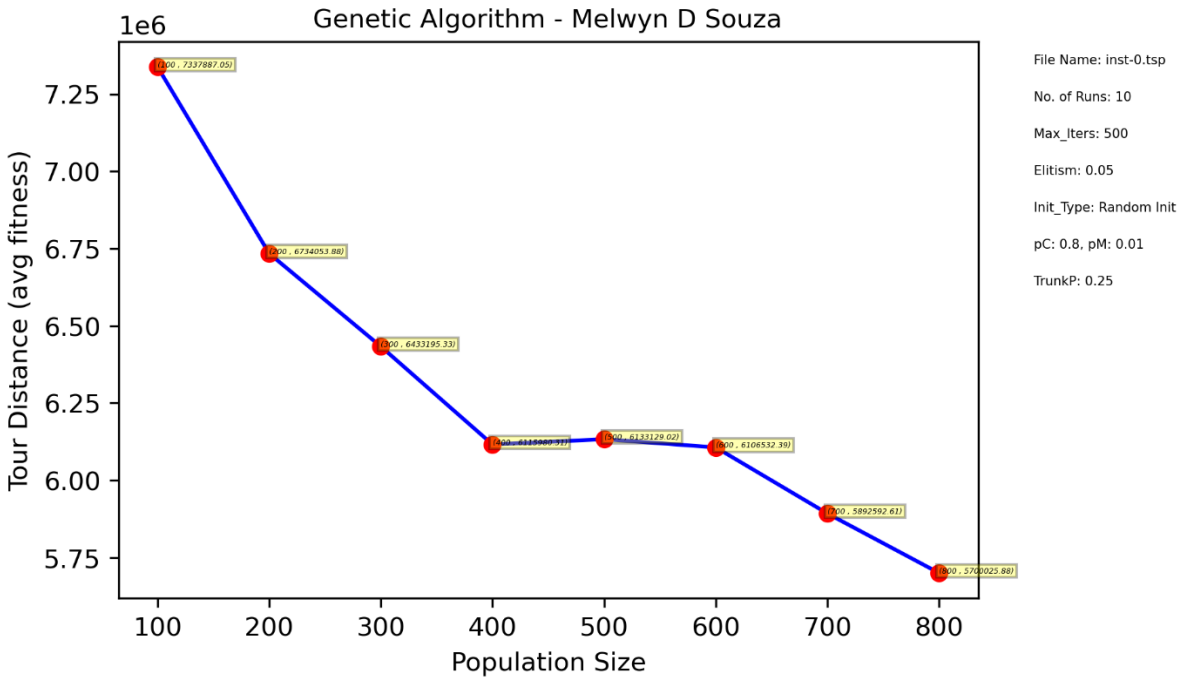
we have high elitism, the new generations will always bounce between the top performers of the previous generations and wont have an opportunity to produce new variations or chromosomes which might be more fitter than the top performers of previous generations, I have put together a graph for better understanding of the impact of elitism, until a certain point, the performance of GA is improved by elitism probability, but when it keeps increasing, the performance drops and this could be because of the convergence of the new generation, for the best results we need best performers from the previous generations and also the new generation has to be diverge

From the graph below, the best results were obtained for elitism probability of 0.05



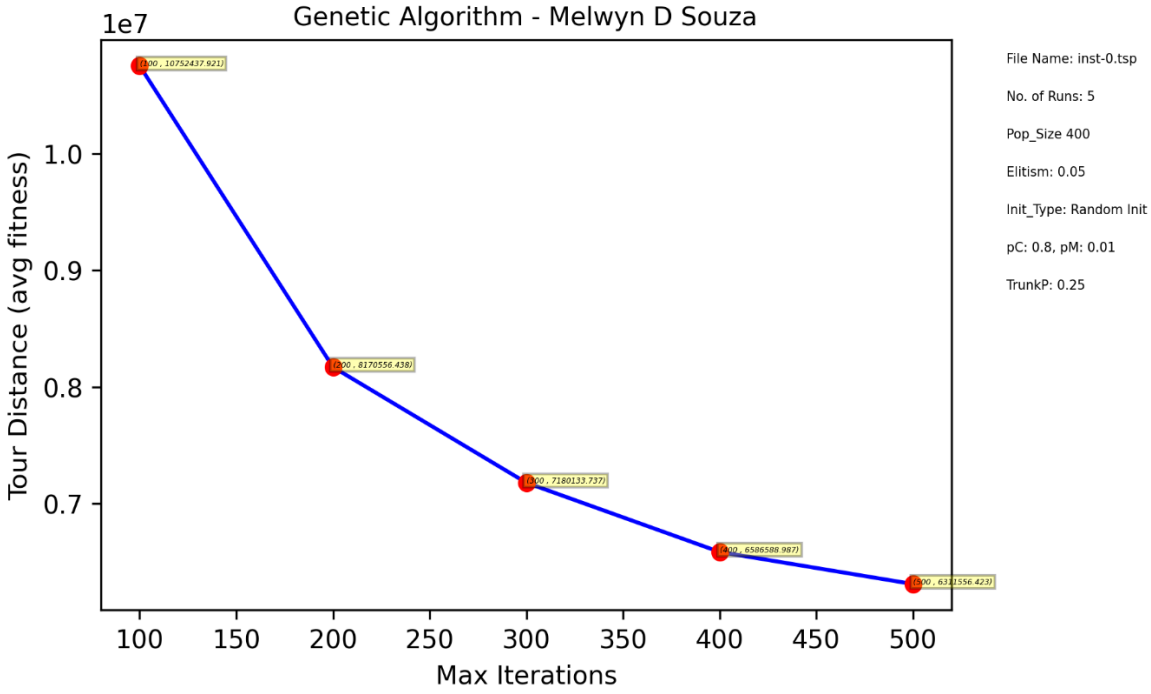
Impact of population size on performance

Bigger the population size, diverge the solutions, since we get a variety of chromosomes or solutions when the population is large, and we select the best performers by our selection methods and elitism, in my opinion its better to have a fairly large population, which will help the GA in finding best solutions, but we have to be careful about the time complexity, if we can attain the same results with half the population selected, we will be wasting space and time with full population size, the impact can be easily understood from my results which I have plotted and displayed below, we can notice that the GA performance increased with the increasing population



Impact of number of iterations on performance

As the number of iterations increase, there's more room for crossover and mutation and selections etc to act and find the better solutions, so on every iteration, if the GA doesn't get stuck in the local minima, then the performance is expected to be improving. We must consider the time complexity when selecting the maximum iterations, was larger iterations will take long time to compute, the graph below shows the visual representation of the impact of number of iterations on the performance of GA



In conclusion, we must be careful in selecting different values for different techniques, I have selected the values for pM, pC, trunkP etc by the results over many iterations, also from the graphs. Genetic Algorithm is helpful in finding the best results as shown and discussed above. For more experimental results, please check out the folder results which is a subdirectory in the submitted zipped folder