Travelling Salesman Problem Using Genetic Algorithms

(A MINOR PROJECT REPORT)

Submitted By

Arijit Ray, Univ. Roll No.-21301217105, Univ. Reg. No.- 172131010016

Bhagyashree Maity, Univ. Roll No.-21301217098, Univ. Reg. No.- 172131010023

Jayashree Mahapatra, Univ. Roll No.-21301217, Univ. Reg. No.- 172131010047

Sankha Misra, Univ. Roll No.-21301217041, Univ. Reg. No.- 17213101080

Under the Supervision of

Prof. Arnab Kole
Assistant Professor

In partial fulfillment for the award of the degree
of
Bachelor of Computer Application

The Heritage Academy
Maulana Abul Kalam Azad University of Technology

2019

ACKNOWLEDGEMENT

We would take the opportunity to thank ***Prof. Gour Banerjee, Principal, The Heritage Academy*** for allowing us to form a group of 4 people and for supporting us with the necessary facilities to make our project worth.

We are thankful to ***Prof. Arnab Kole (Assistant Professor, BCA)*** our ***Project Guide*** who constantly supported us, and ***Prof. Avik Mitra (Assistant Professor, BCA),the Project Coordinator*** for providing and clarifying the administrative formalities related to project proceedings. Their words and instructions have assisted us to excel in our project.

We thank all our other faculty members and technical assistants at The Heritage Academy for paying a significant role during the development of the project. Last but not the least, we thank all our friends for their cooperation and encouragement.

Signature: ________________________ ________

(Arijit Ray)

Signature: ____________________________________

(Bhagyashree Maity)

Signature: ____________________________________

(Jayashree Mahapatra)

Signature: ____________________________________

(Sankha Misra)

The Heritage Academy

Kolkata

PROJECT CERTIFICATE

This is to certify that the following students:

Name of the students	Roll No.	Registration No.
1. **ARIJIT RAY**	21301217105	172131010016
2. **BHAGYASHREE MAITY**	21301217098	172131010023
3. **JAYASHREE MAHAPATRA**	21301217074	172131010047
4. **SANKHA MISRA**	21301217041	172131010080

of 3rd Year 1st Semester in BCA(H) have successfully completed their Minor Project Work on

TRAVELLING SALESMAN PROBLEM USING GENETIC ALGORITHMS

towards *partial fulfillment* of Bachelor of Computer Applications from Maulana Abul Kalam Azad University of Technology, West Bengal in the year **2019.**

Prof. Dr.Gour Banerjee
Principal
The Heritage Academy

Prof. Aranb Kole
Project Guide
Assistant Professor
Department of BCA
The Heritage Academy

Prof. Avik Mitra
Project Coordinator
Assistant Professor
Department of BCA
The Heritage Academy

ABSTRACT

Meta-heuristic Search Algorithms are very efficient optimization techniques when it comes to problems that require large solution space. Instead of giving exact solution, it tries to produce an approximate one that takes less time. These algorithms are problem independent and hence can be applied to a wider variety of problems. Some popular meta-heuristic algorithms in the literature are Genetic Algorithms, Simulated Annealing, Ant Colony Optimization etc.

In this project, we have tried to harness the Genetic Algorithms to solve the travelling salesman problem on ATT48 dataset from TSPLIB and tried to find a solution that has a minimum cost of around 21,000 approx.

CONTENTS

INTRODUCTION

The Travelling Salesman Problem is a classical optimization problem in the field of Computer Science and Operation Research. The formulation of this problem dates back to the 19th century where it was created as a recreational puzzle based on finding a Hamiltonian cycle. According to the problem statement, a salesman must travel between N cities in such a way that it visits each city only once and return to its starting city with the catch that the solution must create a tour with the shortest route possible.

Travelling salesman is a difficult problem to solve. The easiest but the most expensive solution is to simply try all the possible solutions. But for N number of cities the number of solutions will (N-1) be factorial. This means that for only 10 cities there are over 180 thousand combinations. This problem requires special approaches such as Genetic Algorithms, Simulated Annealing, Ant-colony optimisation, etc.

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.

We have used Genetic Algorithm approach to solve this problem. Genetic Algorithm is a meta-heuristic search algorithm. Genetic Algorithm gives optimum results in cases where the search spaces are very huge. It gives a approximate solution which is in general closer to the real solution. If we had to try every possible solution, the number of possible solutions for our dataset is (48-1) factorial. If we assume, that each possible solution takes 1ms, then to compute all of the possible solutions which will take (47 factorial)ms, which is greater than a decade.

The TSP has several applications even in its purest formulation, such as planning, logistics and even manufacturing of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. In many applications, additional constraints such as limited resources or time windows may be imposed.

ABOUT THE DATASET

- Name: ATT48 TSP Dataset
- Source: TSPLIB
- Contains Aerial Co-ordinates of 48 capital of the US

GENETIC ALGORITHMS

A **genetic algorithm** is a search method that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

There are five phases in a genetic algorithm.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

In terms of computation, Genetic Algorithms (GA) is a maximization technique with multiple probable maxima inside the search space. Hence, it may not give always give us the exact solution but will return an approximate one which will be close to the exact solution.

BASIC GA PROCEDURE:

- Choose initial population
- Evaluate the fitness of each individual in the population
- Repeat
    - Select best-ranking individuals to reproduce.
    - Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
    - Evaluate the individual finesses of the offspring
    - Replace worst ranked part of population with offspring
- Until <terminating condition>

BASIC TERMINOLGIES ASSOCIATED WITH GENETIC ALGORITHMS:

- **Chromosomes:**
  A chromosome is a randomly generated one dimensional array of bits, digits/suitable characters that represents a solution in the search space of the problem.
- **Decoding Functions:**
  These functions map a particular chromosome to its solution in the search space.

- **Fitness Functions:**
  Fitness Functions are objective functions that take a candidate solution represented as a chromosome from the search space and evaluates them. As Gas are maximization problems, high fitness values results in better solutions.
- **Population Size:**
  A cluster of chromosomes are called the population. It is a hyper-parameter usually adjusted by the algorithm designer. The initial population is generated at random.
- **Mating Pool:**
  Set of selected chromosomes that will produce new off-springs.
- **GA Operators:**
  The whole GA technique has 3 primary operations:

  1. **Selection:** Selection operator picks chromosomes from the existing population to create a mating pool. There are 2 types of widely used selection techniques:
     - ***Roulette Wheel:*** In this, the chances of the chromosomes to be selected for mating pool are correlated to its fitness function. Imagine a roulette wheel. Now calculate sum of all chromosome finesses in population as *sum_fitness* (*sum_fitness*=$\sum_{i=0}^{popsize} f(chr[i])$*;* where *popsize* is population size; *chr is* chromosomes; *f(x)* is the fitness function*).*

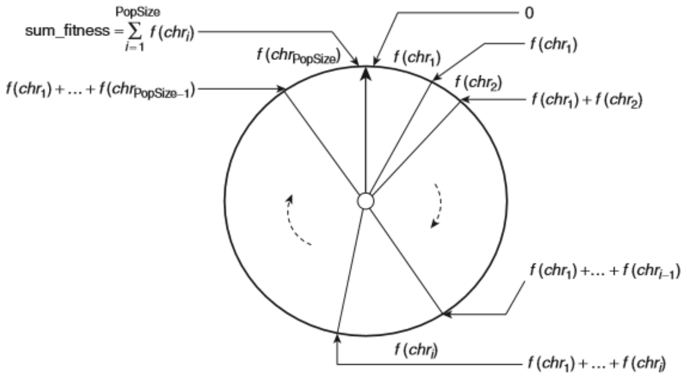

$sum\_fitness = \sum_{i=1}^{PopSize} f(chr_i)$
0
$f(chr_{PopSize})$
$f(chr_1)$
$f(chr_1)$
$f(chr_2)$
$f(chr_1) + f(chr_2)$
$f(chr_1) + \ldots + f(chr_{PopSize-1})$
$f(chr_1) + \ldots + f(chr_{i-1})$
$f(chr_i)$
$f(chr_1) + \ldots + f(chr_i)$

Fig 1: A Roulette Wheel [1]

       This sum fitness is the total circumference of the wheel with partitions equal to *popsize* and the area of partition for each chromosomes is equivalent to its corresponding fitness value. Now, a random number is generated in between 0 and *sum_fitness*, say *d*. The chromosomes within whose limit d falls is taken in for the mating pool. This process is repeated *popsize times* to create a mating pool.
     - ***Tournament Selection:*** In this strategy, 2 chromosomes are random selected from the current population and selected for the mating pool on the basis of their fitness score i.e. whichever has the higher fitness score gets into the mating pool. The procedure runs for *popsize* times.

2. **Crossover:** In this operation, 2 chromosomes are randomly selected and some of their segments are swapped to give 2 new off springs. The swapping operation is done by deciding whether they would go for crossover. For this, a threshold probability p_c is predefined by the algorithm designer. Now, a random number is generated in between 0 and 1, say r. If $r \leq p_c$, the crossover occurs. The crossover point is generated between 0 and *chrlength* (*chrlength* is the length of the chromosomes). The crossover occurs by swapping the segments of the crossover point. There are 2 crossover techniques:
    a. ***One-Point Crossover:*** In this technique, only one crossover point p is generated between 0 and *chrlength* and the bits of the parent chromosomes are swapped from p^{th} bit till *chrlength*
    .

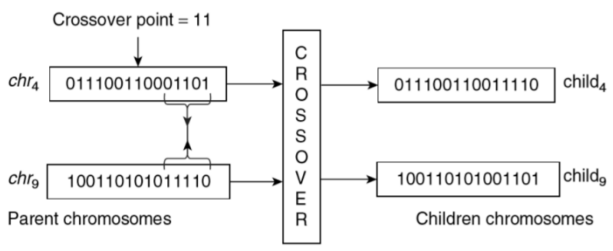


Fig 2: Demo for One-Point Crossover [2]

    b. ***Two-Point Crossover*:** In this technique, 2 crossover points $p_1$and p_2 is generated between 0 and *chrlength* and the bits of the parent chromosomes are swapped from p_1^{th} bit till p_2^{th}bit.

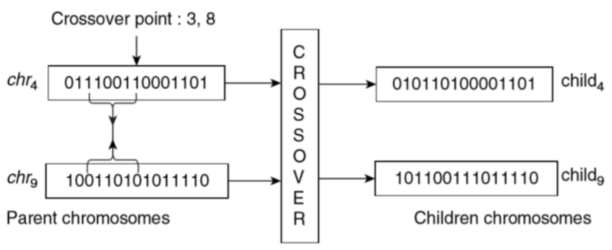


Fig 3: Demo for Two-Point Crossover [3]

3. **Mutation:** The mutation operation is implemented to create a small change the chromosomes so that is explored unbiased. A threshold probability p_m is predefined by the algorithm designer for deciding whether the mutation operation is to be executed or not. Now a random number r between 0 and 1 is generated. If $r \leq p_m$, Mutation occurs. The mutation probability is usually kept low (say $p_m < 0.1$) because mutation is a permanent change in the gene and hence can deviate from the solution.

- **Termination Condition:** Termination can be the no. of iterations or generations of the GA.

PROPOSED ALGORITHM

TRAVELING SALESMAN PROBLEM USING GENETIC ALGORITHMS

ALGORITHM:

- Step-1: Read in the dataset and calculate the aerial distance of all the 48 cities from each other, initialising them into a cost matrix.
- Step-2: Covert the cost matrix into reward matrix.
- Step-3: Initialising 48 binary chromosomes of 38 bit each as initial population
- Step-4: Repeat Step 4 to Step 7 till the number of iterations mentioned by the user
- Step-5: Apply *selection* operation (Tournament/Roulette Wheel) to create a new mating pool
- Step-6: Apply *crossover* and *mutation* operators on the mating pool to generate the *new* population
- Step-7: Calculate fitness score of the new population. Store the elite chromosome i.e. the chromosomes with high fitness value of the new population.
- Step-8: Replace the current population by the new population.
- Step-9: Return the *best* solution of the current population

PARMETERS AND CONFIGURATION

PARAMETERS:

Parameters are customizable units of an algorithm which when modified can impact the performance of the algorithm. The algorithm designer is responsible for formulating and alterations of the parameter as deemed necessary.

In our project, we have considered the following as the necessary parameters of the GA process:

- Number of Iterations
- Crossover Probability
- Mutation Probability

CONFIGURATIONS:

The project was created in DEV C++ (5.11) IDE and run on TDM-GCC 4.9.2 (64 bit Release)

Operating System: Windows 10 Home 64 bit(10.0, Build:18362)

Processor: Intel Core i5-8250U CPU @ 1.6 GHz x 4

RAM: 8 GB

Also Tested and Run on Xcode Version 11.2.1 (11B500)

Operating System: macOS Mojave (Version 10.14.6)

Processor: Intel Core i5-5250U CPU @ 1.6 GHz x 2

RAM: 8 GB

RESULTS AND DISCUSSIONS

RESULT:

Best results were found when threshold crossover probability and mutation probabilities were set to 0.7 and 0.05 respectively. Among the Crossover techniques, 2 point crossover gave better results. Tournament Selection was more effective than Roulette Wheel Selection in our case. As it is a Heuristic Search Algorithm, exact solutions are difficult to find. The whole process is computationally intensive

Population Size for Each Case is 50.

No. of Iterations	Crossover type	Crossover Threshold	Mutation threshold	Result
50000	2-point	0.7	0.05	21951.7
50000	1-point	0.7	0.05	23802.1
75000	2-point	0.7	0.05	22297
100000	2-point	0.7	0.05	23679
100000	2-point	0.7	0.05	22926.2

Table 1: Parameters used for GA

DISCUSSION:

From multiple builds run, we found out that Optimum number of iterations is approximately 50000.

When iterations were greater than 50000, the result was not optimum in majority of the cases

In case of both 1-point and 2-point crossovers, results were similar but 2-point crossover gave results more close to the expected solution. The best result was found using 50000 iterations with 2-point crossover type and the result in 21951.7.

CONCLUSION AND FUTURE WORKS

CONCLUSION

To solve the travelling salesman problem we have used Genetic Algorithm approach. But as we know Genetic Algorithm does not provide us with the exact solution. With every build run, we get a new result. The results are based on random generations and mutations. But, Genetic Algorithm gives us a value close to the actual result. For more accurate results, we need to consider different approaches. Comparing every possible solution is very time consuming and intensive task. Using of meta-heuristic search algorithms and other algorithms are very necessary.

FUTURE SCOPE

The travelling salesman problem has many applications in different fields. To get further better results, we need to make modifications to the proposed algorithm. We might also use different heuristic approaches or different algorithms. We might also build an hybrid algorithm using genetic algorithm and simulated annealing. Ant-colony optimization technique can also be tested for the travelling salesman problem

REFERENCES

1. Fig 12.9, Samir Roy, Udit Chakraborty, “Introduction to soft computing Neuro-fuzzy and Genetic Algorithms”, Chapter 12, Page 11, Pearson India, 2013
2. Fig 12.13, Samir Roy, Udit Chakraborty, “Introduction to soft computing Neuro-fuzzy and Genetic Algorithms”, Chapter 12, Page 14, Pearson India, 2013
3. Fig 12.13, Samir Roy, Udit Chakraborty, “Introduction to soft computing Neuro-fuzzy and Genetic Algorithms”, Chapter 12, Page 14, Pearson India, 2013
4. https://en.wikipedia.org/wiki/Travelling_salesman_problem (Last Accessed: 25-11-19)
5. https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/ (Last Accessed: 26-11-19)

APPENDIX

```
#include<iostream>//including required header files
#include<conio.h>
#include<cstdlib>
#include<ctime>
#include<math.h>
#include<fstream>
using namespace std;
int iter;
double cdef,mdef;
double costMatrix[48][48];
double rewardMatrix[48][48];
int chromosomes[48][288];
int combinations[48][48];//stores decoded chormosomes
double fitness[48];//stores fitness of decoded chromosomes
int cityBuffer[48];//buffer to identify the cities decoded and used for each chromosomes
int elite[100000][48];//stores elite
double elscore[100000];
double cost[100000];
int elcount=0;
int tempchrom[48][288];
int coords[48][2];
int genChrom()//generating binary chromosomes
{
        int i,j;
        for(i=0;i<48;i++)
        {
                for(j=0;j<288;j++)
                {
                        chromosomes[i][j]=(rand()%2);
                }
        }
}
double findDist(int i, int j)// udf to find reward between two cities
{
        double r;
        if(i==j)
                return -1.0;
        else
        {
                r=costMatrix[i][j];
                return r;
```

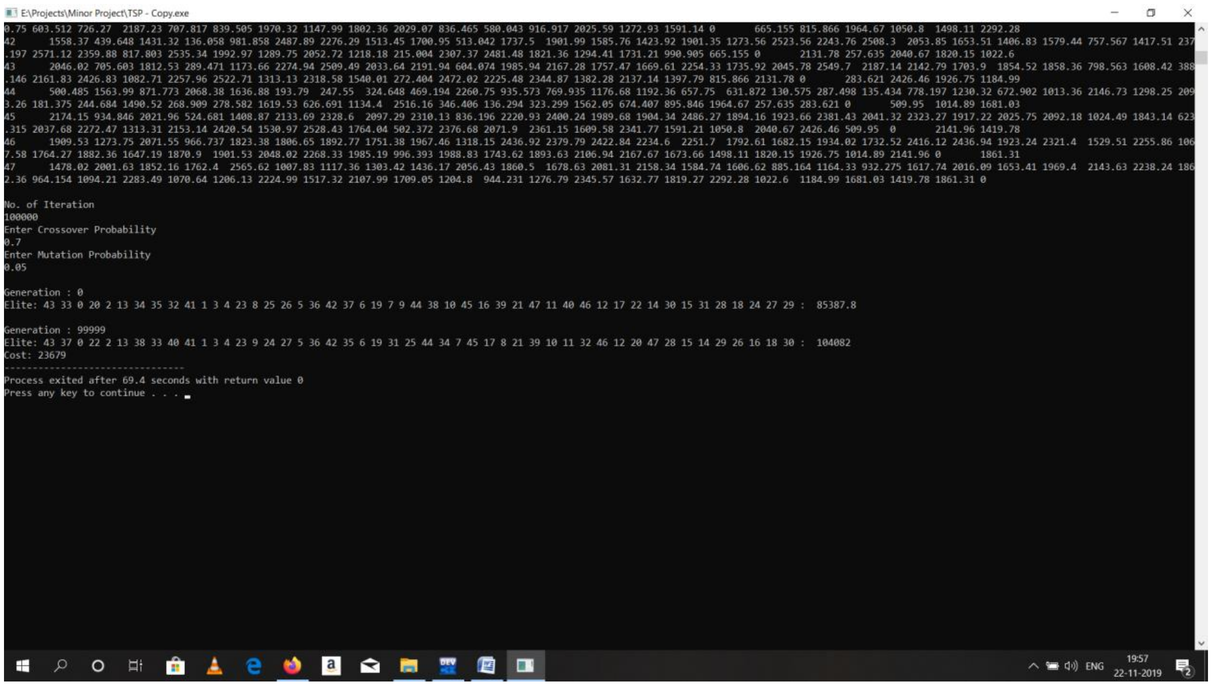
```
	}
}
double dist(int i, int j)
{
	double sum,a,b;
	a=pow((coords[j][0]-coords[i][0]),2);
	b=pow((coords[j][1]-coords[i][1]),2);
	sum=sqrt((a+b)/10.0);
	return sum;
}
int genCost()
{
	int i,j;
	double max=0.0,d=0;
	for(i=0;i<48;i++)
	{
		for(j=i+1;j<48;j++)
		{
			d=dist(i,j);
			costMatrix[i][j]=d;
			costMatrix[j][i]=d;
		}
	}
	for(i=0;i<48;i++)
	{
		for(j=0;j<48;j++)
		{
			if(max<costMatrix[i][j])
				max=costMatrix[i][j];
		}
	}
	cout<<endl<<"Max: "<<max<<endl;
	for(int i=0;i<48;i++)
	{
		for(int j=0;j<48;j++)
		{
			if(i!=j)
				rewardMatrix[i][j]=max-costMatrix[i][j];
		}
	}
}
int printCost()
{
		int i,j,c=0;
		cout<<"Generated Cost Matrix:\n \n";
		for(i=0;i<49;i++)
		{
			for(j=0;j<49;j++)
			{
				if(i==0)
				{
					if(i==0 && j==0)
						cout<<" \t";
					else
						cout<<j-1<<"\t";
				}
				else
				{
					if(j==0)
						cout<<c<<"\t";
					else
						cout<<costMatrix[i-1][j-1]<<"\t";
				}
			}
			if(i>=1)
				++c;
			if(i==0)
				cout<<endl;
			cout<<endl;
		}
	}
	int printReward()
	{
		int i,j,c=0;
		cout<<endl<<"Reward Matrix:\n \n";
		for(i=0;i<49;i++)
		{
			for(j=0;j<49;j++)
			{
				if(i==0)
				{
					if(i==0 && j==0)
						cout<<" \t";
					else
						cout<<j-1<<"\t";
				}
```

```
                        else
                        {
                                if(j==0)
                                        cout<<c<<"\t";
                                else
                                        cout<<rewardMatrix[i-1][j-1]<<"\t";
                        }
                }
                if(i>=1)
                        ++c;
                if(i==0)
                        cout<<endl;
                cout<<endl;
        }
        cout<<endl;
}
double findReward(int i, int j)// udf to find reward between two cities
{
        double r;
        if(i==j)
                return -1.0;
        else
        {
                r=rewardMatrix[i][j];
                return r;
        }
}
void elitism()
{
        int i,ind;
        double max=0;
        for(i=0;i<48;i++)
        {
                if(max<fitness[i])
                {
                        max=fitness[i];
                        ind=i;
                }
        }
        for(i=0;i<48;i++)
                elite[elcount][i]=combinations[ind][i];
        elscore[elcount]=max;
        elcount++;
```

```
}
int decode() //decoding the chromosomes
{
        int i,j,d,c,k;
        for(k=0;k<48;k++)
        {
                c=0;
                for(i=0;i<48;i++)
                {
                        c=i*6;
                        d=0;
                        for(j=5;j>=0;j--)
                        {
                                d=d+(chromosomes[k][c+(5-j)]*pow(2,j));
                        }
                        if(d<48)
                        {
                                while(cityBuffer[d]!=0)
                                        d=(d+1)%48;
                        }
                        else
                        {
                                d=0;
                                while(cityBuffer[d]!=0)
                                        d=(d+1)%48;
                        }
                        cityBuffer[d]=1;
                        combinations[k][i]=d;
                        c++;
                }
                for(i=0;i<48;i++)
                        cityBuffer[i]=0;
        }
}
void fitneScore() //finding and storing fitness scores of each chromosomes
{
        int i,j;
        double r;
        decode();
        for(i=0;i<48;i++)
        {
                r=0.0;
                for(j=0;j<47;j++)
```

```
                        r+=findReward(combinations[i][j],combinations[i][j+1]);
                r+=findReward(combinations[i][47],combinations[i][0]);
                fitness[i]=r;
        }
}
void selection()
{
        int i,j,r1,r2,select;
        for(i=0;i<48;i++)
        {
                r1=rand()%48;
                r2=rand()%48;
                while(r1==r2)
                        r1=rand()%48;
                if(fitness[r1]>fitness[r2])
                        select=r1;
                else
                        select=r2;
                for(j=0;j<288;j++)
                {
                        tempchrom[i][j]=chromosomes[select][j];
                }
        }
        for(i=0;i<48;i++)
        {
                for(j=0;j<288;j++)
                {
                        chromosomes[i][j]=tempchrom[i][j];
                }
        }
}
void crossover()
{
        int i,j,k,swap,pos1,pos2,chr1,chr2;
        double cp;
        for(i=0;i<24;i++)
        {
                cp=(rand()%1000)/1000.0;
                if(cp<cdef)
                {
                        chr1=rand()%48;
                        chr2=rand()%48;
                        while(chr1==chr2)
                            chr1=rand()%48;
                        pos1=rand()%288;
                        pos2=rand()%288;
                        while(pos1==pos2)
                            pos1=rand()%288;
                        if(pos1>pos2)
                        {
                            swap=pos1;
                            pos1=pos2;
                            pos2=swap;
                        }
                        for(k=pos1;k<=pos2;k++)
                        {
                            swap=chromosomes[chr1][i];
                            chromosomes[chr1][i]=chromosomes[chr2][i];
                            chromosomes[chr2][i]=swap;
                        }
                }
        }
}
void mutation()
{
        int i,pos;
        double mp;
        for(i=0;i<48;i++)
        {
                mp=(rand()%1000)/1000.0;
                if(mp<mdef)
                {
                        pos=rand()%288;
                        if(chromosomes[i][pos]==0)
                            chromosomes[i][pos]=1;
                        else
                            chromosomes[i][pos]=0;
                }
        }
}
void GeneticAlgorithm(int n,double c=0.7,double m=0.01)
{
        int i,j;
        iter=n;
        cdef=c;
        mdef=m;
```

```
        genChrom();
        cout<<"Generation : 0"<<endl;
        fitneScore();
        elitism();
        cout<<"Elite: ";
        for(j=0;j<48;j++)
                cout<<elite[elcount-1][j]<<" ";
        cout<<":\t"<<elscore[elcount-1];
        cout<<endl<<endl;
        for(i=1;i<iter;i++)
        {
                if(i%500==0)
                        srand(time(NULL));
                selection();
                crossover();
                mutation();
                fitneScore();
                elitism();
        }
}
int main()
{
        int n;;
        double c,m,dist=0.0;
        ifstream inData("tspdistance.txt");
   for (int i=0; i<48; i++) {
        inData >> coords[i][0] >> coords[i][1]; //Read x and y coordinates at position i
        cout << coords[i][0] <<"\t"<< coords[i][1] << endl; //Output the x and y coordinates
read above
        cityBuffer[i]=0;
        }
   inData.close();
   genCost();
   printCost();
   printReward();
   cout<<"No. of Iteration"<<endl;
   cin>>n;
   cout<<"Enter Crossover Probability"<<endl;
        cin>>c;
        cout<<"Enter Mutation Probability"<<endl;
        cin>>m;
        srand(time(NULL));
        cout<<endl;
        GeneticAlgorithm(n,c,m);
        cout<<"Generation : "<<n-1<<endl;
        cout<<"Elite: ";
        for(int j=0;j<48;j++)
                cout<<elite[elcount-1][j]<<" ";
        cout<<":\t"<<elscore[elcount-1]<<endl;
        cout<<"Cost: ";
        for(int j=0;j<47;j++)
                dist+=findDist(elite[elcount-1][j],elite[elcount-1][j+1]);
        dist+=findDist(elite[elcount-1][47],elite[elcount-1][0]);
        cout<<dist;
    return 0;
}
```



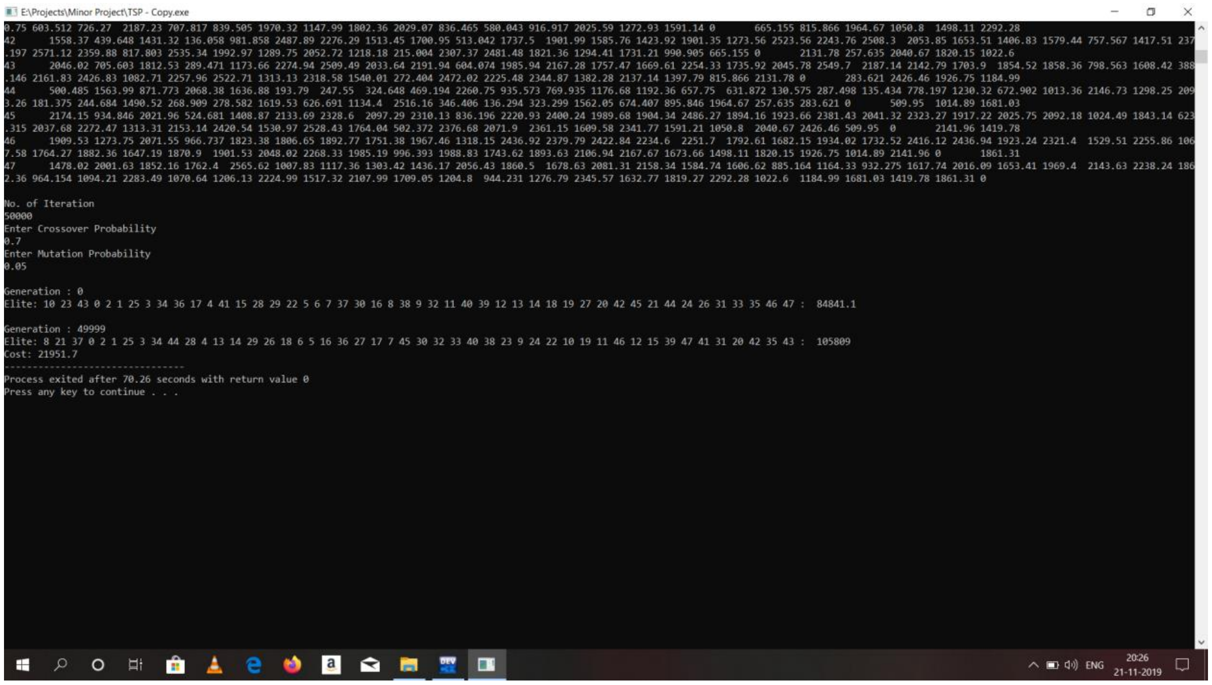
```
E:\Projects\Minor Project\TSP - Copy.exe
0.75 603.512 726.27  2187.23 707.817 839.505 1970.32 1147.99 1802.36 2029.07 836.465 580.043 916.917 2025.59 1272.93 1591.14 0       665.155 815.866 1964.67 1050.8  1498.11 2292.28
42      1558.37 439.648 1431.32 136.058 981.858 2487.89 2276.29 1513.45 1700.95 513.042 1737.5  1901.99 1585.76 1423.92 1901.35 1273.56 2523.56 2243.76 2508.3  2053.85 1653.51 1406.83 1579.44 757.567 1417.51 237
.197 2571.12 2359.88 817.803 2535.34 1992.97 1289.75 2052.72 1218.18 215.004 2307.37 2481.48 1821.36 1294.41 1731.21 990.905 665.155 0       2131.78 257.635 2040.67 1820.15 1022.6
43      2046.02 705.603 1812.53 289.471 1173.66 2274.94 2509.49 2033.64 2191.94 604.074 1985.94 2167.28 1757.47 1669.61 2254.33 1735.92 2045.78 2549.7  2187.14 2142.79 1703.9  1854.52 1858.36 798.563 1608.42 388
.146 2161.83 2426.83 1082.71 2257.96 2522.71 1313.13 2318.58 1540.01 272.404 2472.02 2225.48 2344.87 1382.28 2137.14 1397.79 815.866 2131.78 0       283.621 2426.46 1926.75 1184.99
44      500.485 1563.99 871.773 2068.38 1636.88 193.79  247.55  324.648 469.194 2260.75 935.573 769.935 1176.68 1192.36 657.75  631.872 130.575 287.498 135.434 778.197 1230.32 672.902 1013.36 2146.73 1298.25 209
3.26 181.375 244.684 1490.52 268.909 278.582 1619.53 626.691 1134.4  2516.16 346.406 136.294 323.299 1562.05 674.407 895.846 1964.67 257.635 283.621 0       509.95  1014.89 1681.03
45      2174.15 934.846 2021.96 524.681 1408.87 2133.69 2328.6  2097.29 2310.13 836.196 2220.93 2400.24 1989.68 1904.34 2486.27 1894.16 1923.66 2381.43 2041.32 2323.27 1917.22 2025.75 2092.18 1024.49 1843.14 623
.315 2037.68 2272.47 1313.31 2153.14 2420.54 1530.97 2528.43 1764.04 502.372 2376.68 2071.9  2361.15 1609.58 2341.77 1591.21 1050.8  2040.67 2426.46 509.95  0       2141.96 1419.78
46      1909.53 1273.75 2071.55 966.737 1823.38 1806.65 1892.77 1751.38 1967.46 1318.15 2436.92 2379.79 2422.84 2234.6  2251.7  1792.61 1682.15 1934.02 1732.52 2416.12 2436.94 1923.24 2321.4  1529.51 2255.86 106
7.58 1764.27 1882.36 1647.19 1870.9  1901.53 2048.02 2268.33 1985.19 996.393 1988.83 1743.62 1893.63 2106.94 2167.67 1673.66 1498.11 1820.15 1926.75 1014.89 2141.96 0       1861.31
47      1478.02 2001.63 1852.16 1762.4  2565.62 1007.83 1117.36 1303.42 1436.17 2056.43 1860.5  1678.63 2081.31 2158.34 1584.74 1606.62 885.164 1164.33 932.275 1617.74 2016.09 1653.41 1969.4  2143.63 2238.24 186
2.36 964.154 1094.21 2283.49 1070.64 1206.13 2224.99 1517.32 2107.99 1709.05 1204.8  944.231 1276.79 2345.57 1632.77 1819.27 2292.28 1022.6  1184.99 1681.03 1419.78 1861.31 0

No. of Iteration
100000
Enter Crossover Probability
0.7
Enter Mutation Probability
0.05

Generation : 0
Elite: 43 33 0 20 2 13 34 35 32 41 1 3 4 23 8 25 26 5 36 42 37 6 19 7 9 44 38 10 45 16 39 21 47 11 40 46 12 17 22 14 30 15 31 28 18 24 27 29 :  85387.8

Generation : 99999
Elite: 43 37 0 22 2 13 38 33 40 41 1 3 4 23 9 24 27 5 36 42 35 6 19 31 25 44 34 7 45 17 8 21 39 10 11 32 46 12 20 47 28 15 14 29 26 16 18 30 :  104082
Cost: 23679
--------------------------------
Process exited after 69.4 seconds with return value 0
Press any key to continue . . .
```

Fig 4: Screenshot of an average case



```
0.75 603.512 726.27  2187.23 707.817 839.505 1970.32 1147.99 1802.36 2029.07 836.465 580.043 916.917 2025.59 1272.93 1591.14 0       665.155 815.866 1964.67 1050.8  1498.11 2292.28
42      1558.37 439.648 1431.32 136.058 981.858 2487.89 2276.29 1513.45 1700.95 513.042 1737.5  1901.99 1585.76 1423.92 1901.35 1273.56 2523.56 2243.76 2508.3  2053.85 1653.51 1406.83 1579.44 757.567 1417.51 237
.197 2571.12 2359.88 817.803 2535.34 1992.97 1289.75 2052.72 1218.18 215.004 2307.37 2481.48 1821.36 1294.41 1731.21 990.905 665.155 0       2131.78 257.635 2040.67 1820.15 1022.6
43      2046.02 705.603 1812.53 289.471 1173.66 2274.94 2509.49 2033.64 2191.94 604.074 1985.94 2167.28 1757.47 1669.61 2254.33 1735.92 2045.78 2549.7  2187.14 2142.79 1703.9  1854.52 1858.36 798.563 1608.42 388
.146 2161.83 2426.83 1082.71 2257.96 2522.71 1313.13 2318.58 1540.01 272.404 2472.02 2225.48 2344.87 1382.28 2137.14 1397.79 815.866 2131.78 0       283.621 2426.46 1926.75 1184.99
44      500.485 1563.99 871.773 2068.38 1636.88 193.79  247.55  324.648 469.194 2260.75 935.573 769.935 1176.68 1192.36 657.75  631.872 130.575 287.498 135.434 778.197 1230.32 672.902 1013.36 2146.73 1298.25 209
3.26 181.375 244.684 1490.52 268.909 278.582 1619.53 626.691 1134.4  2516.16 346.406 136.294 323.299 1562.05 674.407 895.846 1964.67 257.635 283.621 0       509.95  1014.89 1681.03
45      2174.15 934.846 2021.96 524.681 1408.87 2133.69 2328.6  2097.29 2310.13 836.196 2220.93 2400.24 1989.68 1904.34 2486.27 1894.16 1923.66 2381.43 2041.32 2323.27 1917.22 2025.75 2092.18 1024.49 1843.14 623
.315 2037.68 2272.47 1313.31 2153.14 2420.54 1530.97 2528.43 1764.04 502.372 2376.68 2071.9  2361.15 1609.58 2341.77 1591.21 1050.8  2040.67 2426.46 509.95  0       2141.96 1419.78
46      1909.53 1273.75 2071.55 966.737 1823.38 1806.65 1892.77 1751.38 1967.46 1318.15 2436.92 2379.79 2422.84 2234.6  2251.7  1792.61 1682.15 1934.02 1732.52 2416.12 2436.94 1923.24 2321.4  1529.51 2255.86 106
7.58 1764.27 1882.36 1647.19 1870.9  1901.53 2048.02 2268.33 1985.19 996.393 1988.83 1743.62 1893.63 2106.94 2167.67 1673.66 1498.11 1820.15 1926.75 1014.89 2141.96 0       1861.31
47      1478.02 2001.63 1852.16 1762.4  2565.62 1007.83 1117.36 1303.42 1436.17 2056.43 1860.5  1678.63 2081.31 2158.34 1584.74 1606.62 885.164 1164.33 932.275 1617.74 2016.09 1653.41 1969.4  2143.63 2238.24 186
2.36 964.154 1094.21 2283.49 1070.64 1206.13 2224.99 1517.32 2107.99 1709.05 1204.8  944.231 1276.79 2345.57 1632.77 1819.27 2292.28 1022.6  1184.99 1681.03 1419.78 1861.31 0

No. of Iteration
50000
Enter Crossover Probability
0.7
Enter Mutation Probability
0.05

Generation : 0
Elite: 10 23 43 0 2 1 25 3 34 36 17 4 41 15 28 29 22 5 6 7 37 30 16 8 38 9 32 11 40 39 12 13 14 18 19 27 20 42 45 21 44 24 26 31 33 35 46 47 :  84841.1

Generation : 49999
Elite: 8 21 37 0 2 1 25 3 34 44 28 4 13 14 29 26 18 6 5 16 36 27 17 7 45 30 32 33 40 38 23 9 24 22 10 19 11 46 12 15 39 47 41 31 20 42 35 43 :  105809
Cost: 21951.7
--------------------------------
Process exited after 70.26 seconds with return value 0
Press any key to continue . . .
```

Fig 5: Screenshot of the best result yielded