TSP USING GENETIC ALGORITHM

N.Moneesh
AM.EN.U4AIE20150
Dept. of Computer Science(AI)
Amrita Vishwa Vidyapeetham
Amritapuri Campus

***Abstract*— The Genetic Algorithm (GA) is a form of evolutionary search strategy for determining the optimum solution to an NP problem. GA requires proper chromosomal expression, as well calculated crossover and mutation operators, to execute an effective search. TSP is a problem in combinatorial optimization. This is an unfinished question, although it is one of the most researched in the subject of optimization. However, as the number of cities grows, so does the problem's complexity. In this papar we will discuss the solving of travelling salesmen problem using genetic algorithm The evolutionary algorithm system's starting population is a matrix comprising estimated distances between cities that a travelling salesman should visit and a randomly chosen city. Then, until the right path is discovered, fresh generations are created. In genetic algorithms, evolutionary biology mechanisms such as inheritance, mutation, selection, and crossover are exploited.**

***Index Terms* - Genetic Algorithm, TSP, Selection, Crossover, Mutation.**

I. INTRODUCTION

A. Travelling Salesman Problem(TSP):

The most famous combinatorial optimization problem is (TSP). The goal of TSP is the shortest path, which is a permutation (or least cost) problem. A TSP can be thought of as an undirected weighted graph with cities as vertices and paths as edges. , the distance of the path is the length of the graph edge. This is a minimization problem that starts and ends at a specific vertex after each vertex has been visited exactly once.

A TSP is used to determine a salesperson's route, starting at home, traveling through a series of cities, and then returning to their original location. Routes have the shortest total travel distance and each city is chosen to be visited only once. This problem is of the difficult type because it cannot be solved in polynomial time. To overcome these problems, several heuristic methods have been created and introduced in the field of operational research. When there are few cities, TSP is easy to solve. However, as the number of cities increases, the solution becomes increasingly difficult due to the large computational time required TSP can be effectively used in various fields of domains, including military and transportation. Because of its flexibility and resilience, the genetic algorithm is another option for solving TSP.

Vehicle routing, computer wiring, wallpaper cutting, and task scheduling are just a few of the unique uses of TSP.

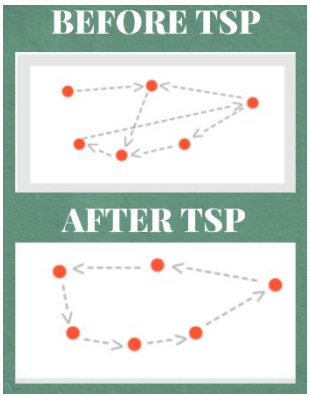

Fig. 1. Tsp example

B. Genetic Algorithm(GA):

John Holland invented the genetic algorithm in the 1970s, but it became famous in the late 1980s. Search and optimization approaches based on Darwin's Principle of Natural Selection are known as genetic algorithms. "Select the best and eliminate the rest," says Darwin's Natural Selection theory. Consider a colony of a certain kind of animal in a forest. Some creatures in the population are more powerful than others, and these features enable them survive in that environment better than others. Assume that the jungle's natural resources, such as water and food, are scarce. As a result, these creatures must compete for resources with one another.

Just only fittest individuals will make it to the end, while the others will perish. GAs is used to address a wide range of issues that are difficult to tackle using other methods.

Evolution, Crossover, and Mutation are the three main processes of a genetic algorithm. By modelling species evolution through natural selection, genetic algorithms execute optimization methodologies. Genetic algorithm begins with a range of problem-solving options that are encoded in the population. A fitness function is used to assess each individual's fitness; a new generation is formed through the selection process; and then crossover and mutation are applied to the newly created new generation. An ideal solution is achieved once the genetic process is terminated. If the termination condition isn't met, the population algorithm is repeated.

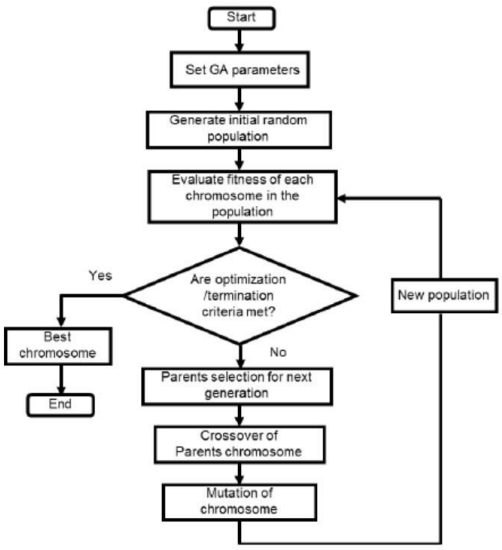

Fig. 2. GA flowchart

II. RELATED WORKS

Varshika proposed a project that leverages the TSP domain and resolves it with the help of genetic algorithmic rule operators. The genetic algorithmic rule's purpose is to make the answer house better.Within the genetic algorithm, the crossover is an essential stage.

Naveen used a variety of genetic algorithmic rule operators to conduct a survey on the disadvantages of travelling salespeople. The proposed study uses a variety of genetic algorithmic rule operators to solve the motion salesman problem.

Omar proposed a novel genetic algorithmic rule in which new crossover operations, population reformulation operations, multi-mutation operations, partial local optimal mutation operations, and arranging operations are used to solve the Traveling Salesman problem.

Chetan solved the problem of the travelling salesman using genetic algorithm operators. The report also presents a comparison of several parent selection methods for the Traveling Salesman problem, such as game equipment, political theory, and Tournament selection methods. This research finds that when the population size is small, all three options provide similar results; however, when the population size is huge, political theory technique provides the best results.

This method main aim is to provide high-quality solutions in a fair length of time. As a result, the sequent Constructive Crossover approach, a brand-new crossover method, is used. This approach can take the best edges from the parent body and create a new offspring with the same edges as the parents or new edges that aren't present in the parent chromosomes.

Kasassbeh proposed a brand-new crossover mechanism called Shared Crossover, which is now being used. This technique is simple and quick, with the primary goal of reducing execution time.

III. METHODOLOGY

Basically in tsp using genetic algorithm we should do 5 steps they are:

- Encoding
- Calculate Fitness Function
- Choosing appropriate crossover operator
- Choosing appropriate selection operator
- Choosing appropriate mutation operator
- Getting efficient path with minimum cost function

1)Encoding:

Encoding is the process of representing chromosomes in a structure, array, list, or other structure using bit, integers, or textual values. The encoding strategy differs depending on the nature of the issue to be solved. Binary and permutation encoding are the two most often utilised methods in encoding. Permutation encoding is what we utilise.

Permutation Encoding: This is useful for problems like the Traveling Salesman Problem (TSP). Every chromosome in TSP is a series of numbers, each of which symbolises a different city to visit.

Because we are performing tsp, we employ permutation encoding in our article.

We assigned a number to each of the six Indian cities of Delhi, Jaipur, Hyderabad, Pune, Haridwar, and Surat. As a result, a path would be represented as a series of numbers ranging from 1-6.

Chromosome 1 1 4 3 2 5 6

Chromosome 2 1 4 2 6 5 3

Chromosome 3 3 6 1 4 2 5

Chromosome 4 6 3 1 5 2 4

Chromosome 5 5 2 6 1 3 4

Chromosome 6 2 6 3 1 5 4

Fig. 3. Encoding example

2)Fitness Function:

The primary goal of the fitness function is to determine if a chromosome is viable. The length of the chromosome is the criterion for a good chromosome in the travelling salesman problem. The overall cost of the journey, as expressed by each chromosome, will be the fitness function. The total of the distance covered in each journey part will be used to compute this. The smaller the sum, the better the approach.

3)Crossover operator:

Crossover is a recombinationoperator that combines the genetic material of two parents to produce offspring.. Crossover is typically used in a genetic algorithm with a high probability in the hopes of producing stronger children by combining the genetic information of two parents. Single point crossover, multi point crossover, and order crossover are the three most prevalent forms of crossover operators.

We employ a single point crossover technique.

Crossover at a single point: A random crossing point along the length of the chromosome is chosen in single point crossover, and the two parents are divided at this point. Following this point, the segments are swapped, resulting in the two offspring.

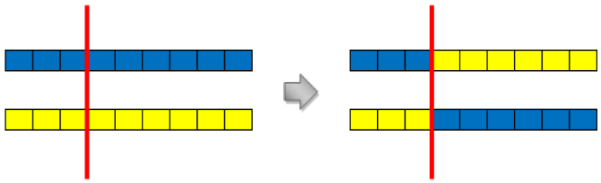

Fig. 4 .Single point crossover example

4)Selection operator:

Selection is the process of selecting two chromosomes from a population to act as parents in order to produce children for the following generation through crossing. The selection approach is focused on selecting the fittest chromosomes in the hopes of producing fitter children. There are three types of selection operators that are often employed they are Roulette Wheel Selection, Rank Selection, Tournament Selection

We deploy roulette wheel selection in this report

One of the most extensively used genetic algorithm selection strategies is roulette wheel selection, which operates as follows. Create a wheel with a diameter equivalent to the total of all chromosomal fitness scores. Each chromosome has a place in this wheel that corresponds to its fitness. The population is then walked through until the goal value is attained, with a random target value selected in the range of fitness sum. The parent chromosome is the one in which the goal value is attained. In our process, more the fitness value of that particular chromosome, then it will have more probability to get selected.

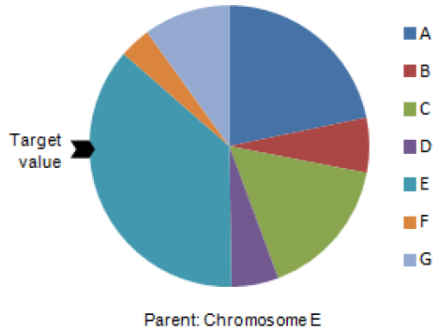

Fig. 5. Single point crossover example

5)Mutation operator:

The purpose of mutation is to introduce new traits into a new population in order to expand the algorithm's search space. The new population is mutated with a very low probability; otherwise, the method is reduced to a random search. Mutation is used to improve population variety by introducing random mutations into the chromosomes. As a result, it avoids the algorithm from becoming stuck in a local minimum.

Swap mutation and displacement mutation are the two most prevalent forms of operators. We have used swap type of mutation operator in our paper..

Swap mutation: 2 genes are selected at random and their positions are swapped.

Fig. 6. Swap mutation example

The best tour will be achieved and the procedure will be ended once a number of operators, such as selection, crossover, and mutation, have been applied to the problem.

IV. RESULTS

Input:

By using random function we will generate the 10 cities(genes), each city will be assigned a integer value between 1 to 10, so 10 cities means there will be 181440 possible routes to travel, Our code will find the 1 best route with least cost.

Output:

Best route: [3, 7, 10, 9, 2, 1, 4, 8, 5, 6]

Graphical representation of traveling the best route:

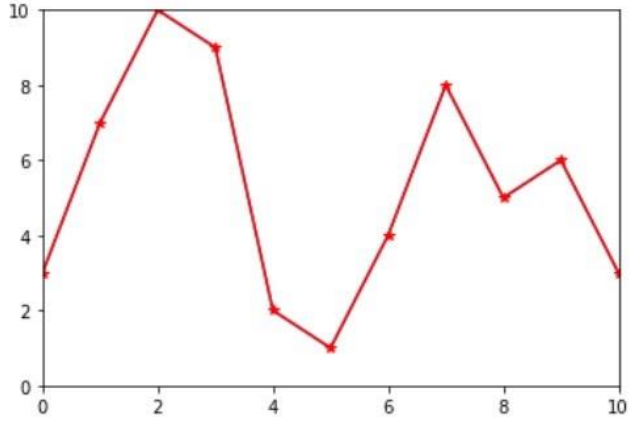

Fig. 7. Route graph

We can see the graph is ending at again (3,3) coordinate because the condition of tsp is that the starting city, should be the ending city so the graph is ending at (3,3) coordinate.

Number of iterations vs Cost function graph:

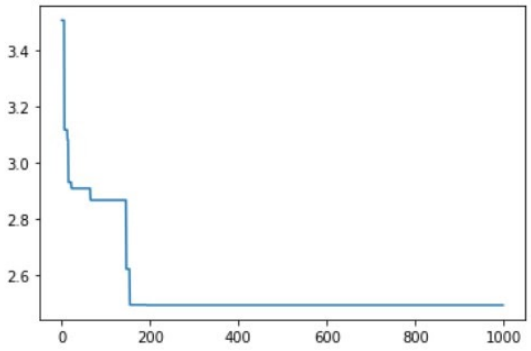

Fig. 8. Cost function graph

We can see x axis is number of iterations , y axis is cost function for each iteration the cost function is reducing as the number of iterations are increasing and at approximately at 100th iteration we can see that the cost function is being constant so we can say that we can stop iterating the code.

V. CONCLUSION

Genetic algorithms use optimization algorithms based on modeling the natural laws of evolution. This is an optimistic way to solve TSP by integrating GA experience. Genetic algorithms find good solutions to a traveling salesman problem depending on how the problem is coded and the crossover and mutation mechanisms used. The goal of this study is to present a highly efficient method for solving TSPs using evolutionary algorithms. In this article, we solved the symmetric TSP, but we will try to solve the asymmetric TSP in the future. Genetic algorithms can be used in a variety of artificial intelligence technologies, as well as more fundamental techniques such as object-oriented programming and robotics.

VI. REFERENCES

[1] M. Lalena, “Travelling Salesman Problem Using Genetic Algorithms”, 2003.

[2] Anand, S., Afreen, N., & Yazdani, S. (2015). A Novel and Efficient Selection Method in Genetic Algorithm. Inetrantional Journal of Computer Applications , 129.

[3] Razali, N. M., & Geraghty, J. (2011). Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. Proceedings of the World Congress on Engineering, II. UK.

[4] Naveen kumar, Karambir and Rajiv Kumar, “A Genetic Algorithm Approach To Study Travelling Salesman Problem”, Journal of Global Research in Computer Science, 2012, Vol. 3.

[5] Dwivedi, TarunaChauhan,SanuSaxena and PrincieAgrawal, “Travelling Salesman Problem using Genetic Algorithm”, International Journal of Computer Applications(IJCA), 2012, pp. 25-30.

[6] Alka Singh Bhagel and Ritesh Rastogi, “Effective Approaches for Solving Large Travelling Salesman Problems with Small Populations”,International Journal of Advances in Engineering Research (IJAER),2011, Vol. (1), Issue (1).

VII. APPENDIX

```
import random
import copy
import math
import matplotlib.pyplot as plt

POPULATION_SIZE = 10
CITIES_SIZE = 10
TOUR_SIZE = 11
NUM_EXECUTION = 999
population = []
x = []
y = []
tour = [[0 for x in range(TOUR_SIZE)] for y in range(TOUR_SIZE)]
dCidade = [[0 for x in range(POPULATION_SIZE)] for y in range(POPULATION_SIZE)]
distances = [0 for x in range(POPULATION_SIZE)]
parentsOne = None
parentsTwo = None
costByExecution = []

def generateFirstPopulation():
    for _ in range(1, POPULATION_SIZE + 1):
        generatePossiblePath()

def generatePossiblePath():
    path = []
    for _ in range(1, CITIES_SIZE + 1):
        randomNum = random.randint(1, 10)
        while(numberExistsInPath(path, randomNum)):
            randomNum = random.randint(1, 10)
        path.append(randomNum)
    population.append(path)

def numberExistsInPath(path, number):
    for i in path:
        if i == number:
            return True
    return False

def generateXandY():
    for _ in range(CITIES_SIZE):
        randomNumber = random.random()
        randomNumber = round(randomNumber, 2)
        x.append(randomNumber)

        randomNumber = random.random()
        randomNumber = round(randomNumber, 2)
        y.append(randomNumber)

def mutate(matrix):
    for i in range(0, len(matrix)):
        for _ in range(0, len(matrix[i])):
            ranNum = random.randint(1, 100)
            if ranNum >= 1 and ranNum <= 5:
                indexOne = random.randint(0, 9)
                indexTwo = random.randint(0, 9)
                auxOne = matrix[i][indexOne]
                auxTwo = matrix[i][indexTwo]
                matrix[i][indexOne] = auxTwo
                matrix[i][indexTwo] = auxOne

def generateTour():
    global tour
    tour = copy.deepcopy(population)
    for ways in tour:
        first = ways[0]
        ways.append(first)

def calculateDistances():
    global distances
    distances = [0 for x in range(POPULATION_SIZE)]
    for i in range(len(population)):
        for j in range(len(population[i])):
            firstPos = 9 if tour[i][j] == 10 else tour[i][j]
            secondPos = 9 if tour[i][j+1] == 10 else tour[i][j+1]
            distances[i] += round(dCidade[firstPos][secondPos], 4)
    dict_dist = {i: distances[i] for i in range(0, len(distances))}
    distances = copy.deepcopy(dict_dist)
    return sorted(distances.items(), key=lambda kv: kv[1])

def fitnessFunction():
    for i in range(len(population)):
        for j in range(len(population)):
            dCidade[i][j] = round(math.sqrt(((x[i] - x[j])**2) + ((y[i] - y[j])**2)), 4)
```

```
    return calculateDistances()

def rouletteFunction(sorted_x):
    global parentsOne
    global parentsTwo
    arr = []
    rouletteArr = []
    for i in range(10):
        arr.append(sorted_x[i][0])
    for j in range(len(arr)):
        for _ in range(10 - j):
            rouletteArr.append(arr[j])
    parentsOne = createParents(rouletteArr)
    parentsTwo = createParents(rouletteArr)

def createParents(rouletteArr):
    parentArr = []
    for _ in range(5):
        parentArr.append(rouletteArr[random.randint(0, 54)])
    return parentArr

def hasDuplicity(auxArray, usedIndexes):
    for i in range(len(auxArray)):
        for j in range(i, len(auxArray)):
            if i != j and auxArray[i] == auxArray[j]:
                if i in usedIndexes:
                    return j
                else:
                    return i
    return -1

def doCycle(sorted_x):
    global population
    children = []

    for i in range(5):
        parentOneAux = parentsOne[i]
        parentTwoAux = parentsTwo[i]
        usedIndexes = []

        randomIndexInsideCromossomus = random.randint(0, POPULATION_SIZE - 1)

        usedIndexes.append(randomIndexInsideCromossomus)

        childOne = copy.deepcopy(population[parentOneAux])
        childTwo = copy.deepcopy(population[parentTwoAux])

        valAuxOne = childOne[randomIndexInsideCromossomus]
        valAuxTwo = childTwo[randomIndexInsideCromossomus]

        childOne[randomIndexInsideCromossomus] = valAuxTwo
        childTwo[randomIndexInsideCromossomus] = valAuxOne

        while(hasDuplicity(childOne, usedIndexes) != -1):
            newIndex = hasDuplicity(childOne, usedIndexes)
            usedIndexes.append(newIndex)

            valAuxOne = childOne[newIndex]
            valAuxTwo = childTwo[newIndex]

            childOne[newIndex] = valAuxTwo
            childTwo[newIndex] = valAuxOne

        children.append(childOne)
        children.append(childTwo)

    mutate(children)

    tempPop = copy.deepcopy(population)

    for i in range(5):
        population[i] = copy.deepcopy(tempPop[sorted_x[i][0]])

    for j in range(5, POPULATION_SIZE):
        population[j] = copy.deepcopy(children[j - 5])


def main():
    generateFirstPopulation()
    generateXandY()
    generateTour()

    for _ in range(NUM_EXECUTION):
        sorted_x = fitnessFunction()
        rouletteFunction(sorted_x)
        doCycle(sorted_x)
        generateTour()
        costByExecution.append(sorted_x[0][1])
    sorted_x = fitnessFunction()

    print('Total population: %s' %(POPULATION_SIZE))
    print('Mutation probability: 5%')
    print('Number Of cities: %s' %(CITIES_SIZE))
    print('Optimal path cost: %s' %sorted_x[0][1])
    print('Best route: %s' %population[0])

    plt.plot(tour[0])
    plt.plot(tour[0], 'ro')
    plt.axis([0, 20, 0, 20])
    plt.show()

    plt.plot(costByExecution)
    plt.show()

if __name__ == "__main__":
    main()
```