

1.1 Overview

A path connects nodes in a network represented by a graph. In a network, the packets are transferred from source to destination through different paths. This optimal selection of nodes will enable to achieve a high performance in the network. There may be a number of paths existing between any two nodes (source and destination), and the best path in terms of cost, is known as the shortest path (SP). The Shortest path problem uses an algorithm, to obtain the shortest path in a given network. SP problems are used for various applications such as in speech recognition, transportation, communications, and various other fields. A number of algorithms such as Bellman-Ford, Dijkstra's algorithm, are present for the SP problem, but there are different drawbacks associated with it [6]. These shortcomings hence have given rise to several other SP algorithms. Genetic algorithm (GA) is an evolutionary computation technique which considers chromosomes as path and genes as nodes in the path [2]. Other than genetic algorithm, we have particle swarm optimization technique for finding shortest path in a network. Particle Swarm Optimization (PSO) algorithm considers different paths between source and destination as particles [5]. Here we have solved Travelling Salesman Problem using PSO and GA separately and compared the results obtained by both the algorithms.

A good shortest path algorithm should be capable of finding optimal path in less time and be simply with less complexity. It must have low overhead, stable and must remain flexible. Different type of routing algorithms is there which have been made for specific kind of network as well as for different routing purpose. But there are very few algorithms which uses intelligent methods for path finding. With the increasing internet usage these basic routing approach will not serve the purpose of faster routing.

The aspect of such intelligence has been dealt in cognitive network and autonomic networking. Intelligent network was well formulated based on heuristic algorithm, evolutionary algorithm like PSO and GA and human immune system. Applying artificial intelligent methods in networks and gaining higher efficiency in operation have a greater significance in the area of optimal

communication, realizing best QoS and security, the management of different layers of protocol and development of application software [8].

These algorithms are used for various applications depending upon their specifications and requirements. But the current trend in the usage of the internet shows that these protocols will not be self-sustainable for meeting the requirement for the fast delivery of packets. Several algorithms have been implemented and analyzed for SP problems. Dijkstra's algorithm is used for solving shortest path problem but only for non-negative weighted networks. For solving network with negative weights, Bellman-Ford algorithm is used. Also it solves the shortest path problem for all pairs of shortest path. The algorithms mentioned above are simple and very easy to use but they have various drawbacks which make them unsuitable for different applications.

The PSO is a method of optimizing the solution by continuously trying to improve the previous or intermediate solution and thus getting the final solution. Simulation studies have been carried out which shows that these optimization methods have very few constraints. In the implementation of such algorithm the cost involved is less and hence they are used for real-time environment. The PSO and genetic algorithm (GA) have been compared and it is inferred that PSO works faster than GA to find the shortest path in a network in less time [2].

1.2 Particle Swarm Optimization (PSO)

PSO is an optimization technique developed by Kennedy and Eberhart, and is inspired by the social behavior of bird flock [5]. When a group of birds moves through certain distance the position of each bird changes due to change in velocity. The flock of birds updates their position according to the velocity and position of the bird leading them. Hence each and every bird moves closer to the bird which is nearest to the food [6].

PSO is a population-based technique, inspired by the social behavior of particles inside a group of particles known as swarm for example flock of birds. In addition PSO is a collection of particles in an N-dimensional space. The rules for how the particles will move through the space depends upon the simple natural flocking rules that cause the particle to set its position around the best-found particle in the hope of finding a better particle. So particles will usually looking forward for the optimum solution. PSO is a simple algorithm, fast, effective, and can be used for

different types of optimization problems.

The concept of PSO has been extended to different applications like biomedical applications, classification and clustering, communication networks, combinatorial optimization, and many other areas. PSO gives us the optimal solution which may or may not be the best solution.

PSO is used to maximize or minimize fitness function by updating the velocity after every loop. Particles (path) obtained by some random function act as an input to PSO algorithm. Our main objective is to find the particle having minimum cost.

PSO optimizes the certain particles (a set of paths in a network) for some number of iterations, by considering parameters such as personal best (pBest) of particle and velocity. The formulae we have used to calculate velocity and position is specified in the algorithm discussed in the method used section.

PBest or personal best is the best position of the particle. And gBest or global best is the best position of all the particles when compared. After every iteration, the personal Best of particle is calculated which is used to calculate the global best value which is the shortest path in the network. The pBest value is calculated which depends on the value of fitness of each particle in every loop. The fitness value obtained for the particle is the indication of the best path to be considered for packet transmission [8].

1.3 Genetic Algorithm

Genetic algorithm has a major application in route selection algorithms. Genetic algorithm comes under the class of evolutionary algorithm. GA considers a population of chromosomes that represent the possible paths between a source and destination. A fitness function is used to evaluate the developments which take place in the population; chromosomes are made to reproduce by using crossover and mutation techniques so as to generate new offspring which provides better solution. This process is repeated for a number of times until an optimal solution is obtained [9].

GA is applied on the population of chromosomes in parallel which helps GA to explore the complete problem space in all direction. GA is best for the problems where solution space is

huge and time taken to search is exhaustive. It does not need to store any previous knowledge to obtain the solution.

In both PSO and GA a fitness function is used which calculates the fitness value of particle and chromosome respectively depending upon some factor like bandwidth. For GA crossovers and mutations techniques are used to generate different possible chromosomes. For PSO the priority of node in each iteration is checked depending upon which position and velocity of a particle is modified. At the end of each iteration the pBest of each particle is been updated and at the final iteration the best particle pBest becomes gBest which gives the shortest path from source to destination [1].

The crossover operation is applied on chromosomes of equal length. Crossover techniques like 1-point crossover technique and 2-point crossover technique are used for crossover.

The example on application of 1-point crossover and 2-point crossover technique is given below.

Crossover example:

A. 1-point example:

Parent1: 1, 2, 3, 4, 5, 6, 7, 8

Parent2: 1, 1, 3, 3, 4, 5, 7, 8

Random choice: $k = 5$

Child: 1, 2, 3, 4, 4, 6, 7, 8

Child: 1, 1, 3, 3, 5, 5, 7, 8

B. 2-point example:

Parent1: 1, 2, 3, 4, 5, 6, 7, 8

Parent2: 1, 1, 3, 3, 4, 5, 7, 8

Random choices: $j = 4, k = 6$

Child: 1, 2, 3, 3, 5, 5, 7, 8

Child: 1, 1, 3, 4, 5, 6, 7, 8

Mutation example:

a. Swap mutation example:

Parent: 1, 2, 3, 4, 5, 6, 7, 8

Random choices: $i = 3$; $j = 6$

Child: 1, 2, 6, 4, 5, 3, 7, 8

b. Adjacent-swap mutation example

Parent: 1, 2, 3, 4, 5, 6, 7, 8

Random choice: $j = 6$

Child: 1, 2, 3, 4, 5, 7, 6, 8

1.4 Motivation

A network consists of different nodes which are interconnected to each other and packets are transferred from source node to destination node through a network of interconnected communication link. The nodes through which packets are transferred may be so much congested and there may exist so many paths between those two nodes so we require efficient methods to find out the optimal path ensuring robust data communication.

Particle swarm optimization and genetic algorithm is the efficient method for this purpose. The reason why we have chosen these algorithms is that PSO is an important swarm intelligent algorithm with fast convergence speed and easy implementation. PSO makes use of cognitive and social information among the individual particles to find an optimal solution. Genetic algorithm is a tree based representation and manipulation and it find the optimal solution by the process of simulated evolution by employing the biological theory of genetics and the Darwinian principle of survival of fittest.

1.5 Problem Statement and Objectives

The main objective of this proposed work is to apply Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) to solve Travelling Salesman Problem and compare the results obtained by both of them which can be useful for finding a shortest path in a network in less time.

The work done is basically divided into two parts:

- (1) Firstly, we have applied PSO on the TSP and recorded and compared the result with different particle count i.e. no of routes we have considered and maximum velocity i.e. maximum change allowed for each particle.
- (2) Secondly, we have applied GA on the TSP and recorded the results by generating new offspring from the existing population of chromosomes by using crossover and mutation techniques.

Finally, we compared the results obtained by the application of PSO and GA on the TSP.

1.6 Organization of the Report

The report is divided into 5 chapters. Explanation about each chapter is given below.

Introduction chapter gives an overview about the routing problems and a brief introduction about TSP, PSO and GA.

Literature Survey contains introduction about the basic idea about Particle Swarm Optimization, Genetic Algorithm, and Travelling Salesman Problem. It also explains how PSO and GA can be applied on TSP

Proposed Method chapter contains the methodology used for the application of PSO and GA for solving TSP

Experimental Result and Analysis chapter contains the result obtained on application of PSO and GA separately. The results obtained are represented using two dimensional spaces as a coordinates for cities. The results obtained for PSO and GA are then compared.

Conclusion and Future Work chapter gives overall concluded results which shows that PSO works faster than GA.

2.1 Travelling Salesman Problem

The Travelling Salesman Problem is one of the most used problem in the field of computer science which describes about a salesman who needs to travel between N cities keeping in mind that the salesman visits each city exactly once and finishes where he started the trip. Every city is connected to one or more adjacent cities. In a graphical form we can represent these cities in the form of nodes. Each of these cities is connected by an edge which has some weight associated with it. The weight or cost describes how difficult it is to traverse the edge on the graph, and may be given for e.g. by the cost of a ticket of airplane or train, or by the distance between the two cities, or time required to complete the distance between the two cities. The aim of the salesman is to find the shortest path so as to minimize the cost of the travelling all cities. The Travelling Salesman Problem is a very important optimization problem that has intrigued computer scientists and mathematicians. It has various applications in science and engineering. For example, in the manufacture of a circuit board, it is important to determine the best possible order in which a laser would drill numerous holes. A good solution to this problem reduces production costs for the manufacturer.

The Traveling Salesman problem (TSP) is one of the mostly used and old problems in Computer Science and Operations Research. It can be stated as:

A network with n nodes (or cities), with node 1 as headquarters and a travel cost (or distance, or travel time etc.,) matrix $C = [c_{ij}]$ of order n associated with ordered node pairs (i, j) is given. The problem is to find a least cost Hamiltonian cycle [3].

On the basis of cost matrix, the TSPs are classified mainly into two groups – symmetric and asymmetric [4]. The TSP is symmetric if $c_{ij} = c_{ji}$, i, j and asymmetric otherwise. For an n -city asymmetric TSP, there are $(n-1)!$ possible solutions.

In either case (symmetric or asymmetric) the number of solutions becomes extremely large for even not so large n so that the search is impracticable.

There are mainly three reasons why TSP has been attention seeker of many researchers and is a very important research area. First, a large number of real-world problems can be compared with TSP. Second, it was proved to be NP- Complete problem. Third, NP-Complete problems are intractable as no one has ever found any really efficient way of solving them for large problem size. Also, NP-Complete problems are known to be more or less equivalent to each other if one knew how to solve one of them one could solve the lot.

The methods that provide the exact optimal solution to the problem are called exact methods. A basic way of solving the TSP is simply to list all the possible solutions, evaluate their cost values and pick out the best possible solution. However it is obvious that this search is exhaustive, inefficient and impracticable because of a large number of possible solutions to the TSP even for problem of normal size. Since practical applications require solving problems with large size, hence emphasis is to find exactly optimal solutions to TSP which are heuristic, efficient and provide solutions in reasonable time and establishing the degree of goodness. Genetic algorithm (GA) is one of the heuristic algorithms that have been used widely to solve the TSP.

TSP provides a good platform for testing optimization techniques, many researchers in various fields like artificial intelligence, mathematics, physics, and biology. Researchers devote themselves to find the most efficient methods for solving the TSP using algorithms such as genetic algorithms (GAs), ant colony optimization (ACO), simulated annealing (SA), neural networks (NN), particle swarm optimization (PSO), evolutionary algorithms (EA), mimetic computing, etc. [14]. Besides, there are various practical applications of the TSP in the real world such as vehicle routing, data association, data transmission in computer networks, job scheduling, DNA sequencing, drilling of printed circuits boards, clustering of data arrays, image processing and pattern recognition, analysis of the structure of crystals, transportation and logistics [15].

2.1.1 Hamiltonian Path

A Hamiltonian path is a path in which each vertex is visited exactly once is known as Hamiltonian path. A graph which contains a Hamiltonian path is called a traceable graph. A graph is said to be Hamiltonian-connected if there exists a Hamiltonian path for every pair of vertices.

A Hamiltonian cycle, Hamiltonian circuit, vertex tour or graph cycle is a cycle in which each vertex is visited exactly once (except for the vertex that is both the source and destination, which is visited twice). A graph containing a Hamiltonian cycle is known as Hamiltonian graph.

Directed graphs also have Hamiltonian path in which each edge of a path or cycle can only be traced in a single direction.

A Hamiltonian path can be converted into Hamiltonian circuit by using edge decomposition which converts it into a Hamiltonian circuit. A Hamiltonian cycle can be converted to a Hamiltonian path by removing one of its edges, but a Hamiltonian path can be converted to Hamiltonian cycle by connecting its adjacent end points. All Hamiltonian graphs are biconnected, but it is not necessary that biconnected graph is Hamiltonian [25].

An Eulerian graph G is a graph in which every vertex has even degree. An Eulerian graph necessarily has an Euler tour in which there is a closed walk which passes through each edge of G exactly once. So, Euler tour corresponds to a Hamiltonian cycle in the line graph $L(G)$ and it can be said that line graph is Hamiltonian graph. Line graphs may have other Hamiltonian cycles that do not correspond to Euler tours, and in particular the line graph $L(G)$ of every Hamiltonian graph G is itself Hamiltonian, regardless of whether the graph G is Eulerian [24]. A tournament (with more than two vertices) is Hamiltonian if and only if it is strongly connected. The number of different Hamiltonian cycles in a complete undirected graph on n vertices is $(n - 1)! / 2$ and in a complete directed graph on n vertices is $(n - 1)!$ [25].

2.2 Genetic Algorithm

Genetic algorithms (GAs) deals with the concept of the survival of the fittest among the species which are generated by randomly changing the genes in the chromosomes in the evolutionary

biological species. To solve some real life problem using GA, two of the main requirements are to be satisfied

(1) A string known as a chromosome in terms of GA which can represent a solution of the solution space, and

(2) A fitness function is used to measure the goodness of a solution

A simple GA works by randomly generating population of chromosome, which can be referred as gene pool and then applying different operators to create new and better populations as their successive generations. The first operator is reproduction where chromosome is copied to the next generation with the probability based on their fitness function value [12]. The second operator is crossover where random pairs are selected and mated to create new chromosome. The third operator, mutation, which is a random alteration of the gene position in the chromosome. The reproduction together with crossover is the most powerful process in the GA .

Mutation diversifies the search space and protects from loss of genetic material that can be caused by reproduction and crossover [9]. So, the probability of applying mutation is set to be very low, whereas the probability of crossover is set to be very high.

2.2.1 Genetic Coding

To apply GA for any optimization problem, the most important step is encoding solutions as feasible chromosomes so that the crossovers of feasible chromosomes result in more feasible chromosomes. The techniques for encoding solutions require special care, a good intelligence and depend on the problem statement. For the TSP, solution is typically represented by chromosome of length equal to the number of nodes in the problem. Each gene of a chromosome represents a node such that no node can appear twice in the same chromosome. There are mainly two mostly used representation methods for representing tour of the TSP – path representation and adjacency representation. The path representation for a tour simply lists the label of nodes. For example, let $\{1, 2, 3, 4, 5\}$ be the labels of nodes in a 5 node instance, then a tour $\{1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 1\}$ may be represented as (1, 3, 4, 2, 5) [11].

2.2.2 Fitness Function

The GAs are used for maximization problem. For minimization problem, one way of defining a fitness functions $F(x) = 1/f(x)$ where $f(x)$ is a objective function. As TSP is a minimization problem, we consider this fitness function $F(x) = 1/f(x)$ where $f(x)$ calculates cost of the tour represented by a chromosome.

2.2.3 Reproduction Operator

Reproduction/Selection process is a way in which chromosomes are copied into next generation mating pool according to the probability associated with their fitness value. When we assign the higher portion of the highly fit chromosomes to the next generation, reproduction mimics the Darwinian survival-of-the-fittest in the natural world. In natural population, fitness is determined by a creature's ability to survive predators, pestilence, and other obstacles to adulthood and subsequent reproduction. No new chromosome is produced in this phase. The reproduction operator which is commonly used is the proportionate reproduction operator, where a string is selected for the mating pool with a probability proportional to its fitness value.

2.2.4 Sequential Constructive Crossover Operator (SCX)

The search space for the solution is generated by creating new chromosomes from existing ones. Crossover is the most important search process. Different crossover techniques are used to generate new children like one-point crossover, two-point crossover. Initially in a crossover a pair of parents is selected from the pool of chromosome randomly. In second step, a point usually called crossover site is selected randomly, and the information of the two parent chromosome after the crossover site are interchanged resulting into two new offspring.

The algorithm for the SCX is as follows:

Step 1: - Start from node 1 (i.e., current node $p=1$).

Step 2: - Sequentially search both of the parent chromosomes and consider the first legitimate node (the node that is not yet visited) appeared after node p in each parent. If no legitimate node after node p is present in any of the parent, search sequentially the nodes $\{2, 3, \dots, n\}$ and consider the first legitimate node, and go to Step 3.

Step 3: Suppose the node α and the node β are found in 1st and 2nd parent respectively, then for selecting the next node go to Step 4.

Step 4: If $cp\alpha < cp\beta$, then select node α , otherwise, node β as the next node and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, then stop, otherwise, rename the present node as node p and go to Step 2.

Let us illustrate the SCX through the example given as cost matrix in Table 2.1. Let a pair of selected chromosomes be P1: (1, 5, 7, 3, 6, 4, 2) and P2: (1, 6, 2, 4, 3, 5, 7) with values 312 and 331 respectively [12].

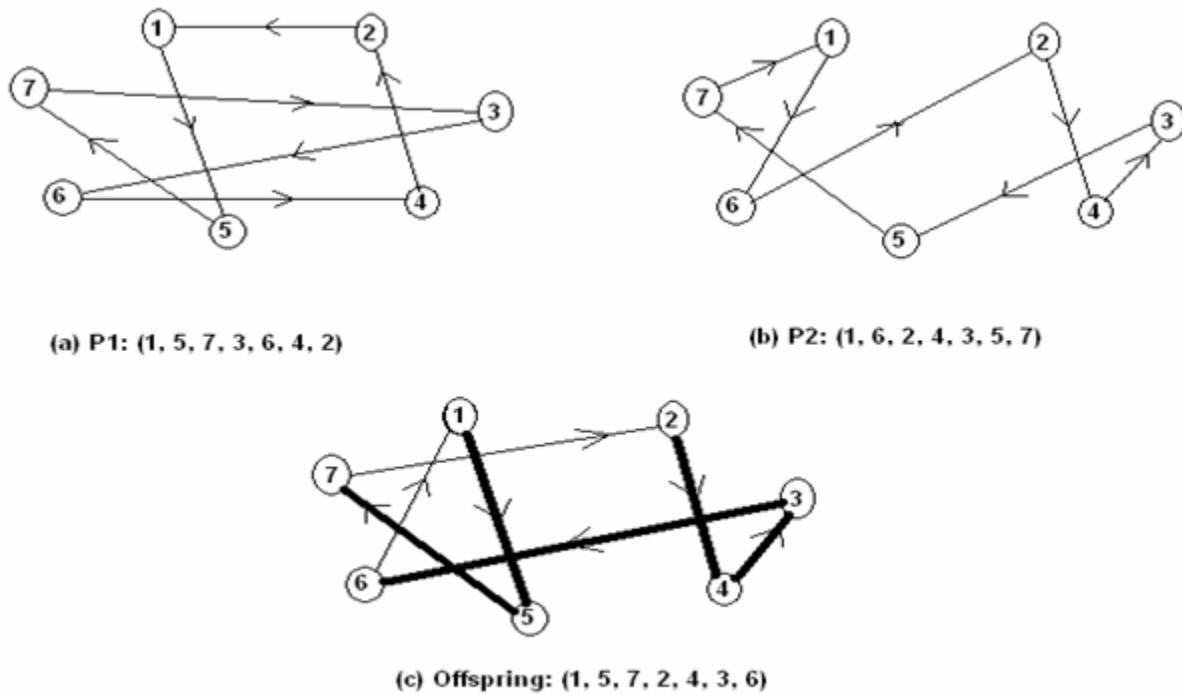


Figure 2.1 : Example of Sequential Constructive Crossover Operator

Table 2.1: The Cost Matrix

Node	1	2	3	4	5	6	7
1	999	75	99	9	35	63	8
2	51	999	86	46	88	29	20
3	100	5	999	16	28	35	28
4	20	45	11	999	59	53	49
5	86	63	33	65	999	76	72
6	36	53	89	31	21	999	52
7	58	31	43	67	52	60	999

2.2.5 Offspring by Two Other Crossover Operators

The other crossover operators used for generating offspring are – edge recombination crossover (ERX) and generalized N-point crossover (GNX) for producing offspring which uses same pair of parents P1 and P2 as used in SCX.

For ERX, the edge table of the example discussed above is shown in Table 2.2 [12]. The new offspring is initialized with node 1, 5, 2, 6 and 7 are the candidates for the next node. Node 6 has three edges and therefore not considered. Assume node 5 is randomly chosen. Node 5 is connected to nodes 7 and 3 so node 7 is chosen next. Node 7 only has an edge to node 3 so node 3 is chosen next. Node 3 has edges to nodes 4 and 6 both of which have two edges. Suppose the node 4 is randomly chosen then the node 4 has edges to the nodes 6 and 2 both of which have one edge left. Next randomly choose node 6 which has an edge to node 2 of course this is the last node to be selected, so node 2 is chosen next. The resulting offspring may be (1, 5, 7, 3, 4, 6, 2).

Table 2.2: Edge Table for the Parents P1 and P2

Node	Edge list	Node	Edge list
1	5, 2, 6, 7	5	1, 7, 3
2	4,1,6	6	3,4,1,2
3	7, 6, 4, 5	7	5, 3, 1

For GNX, let's take $N=2$ and consider same parents P1: (1, 5, 7, 3, 6, 4, 2) and P2: (1, 6, 2, 4, 3, 5, 7) and G2X with cross points 3 and 5. Suppose the order in which the segments are tested is (2, 3, 1). Then the 2nd Segment of P will be inserted whole, giving the proto-child (x, x, x, 3, 6, x, x). Nodes in the 3rd segment from P2 will then be tested in a random order. Both the city 5 and 7 will be accepted, giving segment of P will be inserted whole, giving the proto-child (x, x, x, 3, 6, x, x). Nodes in the 3rd segment from p2 will be tested in a random order. So the final child may be given by (1, 4, 2, 3, 6, 5, 7) [12].

2.2.6 Survivor Selection

Survivor selection method is used for selecting population for next generating after the crossover operation. This survivor selection method of GA considers two kinds of chromosomes for the next generation: (1) parents in current population of size n , and (2) offspring that are generated by crossover of size n . The survivor selection method combines chromosomes in (1) and (2), sorts them in ascending order according to their fitness, and selects the first m chromosomes for the next generation. In worst case, all the parents which were used in present generation will survive and move to the next generation.

2.2.7 Mutation Operator

The mutation operator randomly selects a gene and changes that gene with another gene, thus modifying information. Mutation is basically used because without mutation some aspect of genetic material could be lost forever as less fit chromosome are discarded in next generation from the fact that as the less fit members of successive generations are discarded; some aspects of genetic material could be lost forever. By performing occasional random changes in the chromosomes, mutation ensure that new parts of the search space are reached, which reproduction and crossover alone couldn't fully guarantee. In doing so mutation ensures that no important features are completely lost, thus maintaining the diversity. For the TSP, the classical mutation operator does not work [10]. For this investigation, we have considered the reciprocal exchange mutation that selects two nodes randomly and swaps them.

2.2.8 Control Parameters

These are the parameters that govern the GA search process. Some of them are:

Population size: - It determines the total number of chromosome in the search space.

Crossover probability: - It specifies the probability of crossover occurring between two chromosomes.

Mutation probability: - It specifies the probability of doing mutation.

Termination criteria: - It specifies when to terminate the genetic search.

2.3 Particle Swarm Optimization (PSO) for Network Optimization

One of the inspiration from which PSO was developed was bird flocks which considers birds as a particle moving in a flock chasing for food. Each bird has some velocity and they continuously change their positions to get nearest to the food. Basically, particle swarms is used as a means to simulate social behavior of bird flocks by using computer-based model.

Swarm intelligence is an important method based on the collective behavior of decentralized and self-organized systems in computational intelligence [6]. It consists of a population which

simulates the behavior of birds in the real world. There are many swarm intelligence optimization algorithms, such as particle swarm optimization, genetic algorithms, bee colony algorithm, ant colony optimization, differential evolution, fish -warm algorithm, etc. Due to the easy implementation and simple concept, PSO has gained much attention and been successfully applied in a variety of fields especially for optimization problems.

Goldberg developed a PSO algorithm for the TSP where the idea of distinct velocity operators is introduced. The velocity operators are defined according to the possible movements a particle is allowed to do. This algorithmic proposal obtained very promising results [4].

2.3.1 Classical PSO

The basic principles of PSO are very simple. A set of moving particles is initially thrown inside the search space. Each of these particles has the following features:

- (1) It has a position and a velocity
- (2) It knows its position
- (3) It knows its neighbors and best previous position(personal best)

At each step, the particle can move in three different directions as described below:

- (1) To follow its own way
- (2) To go towards its best previous position
- (3) To go towards the best neighbor's best previous position, or towards the best neighbor

The decision depends upon three parameters:

- (1) how much the particle trusts itself
- (2) how much it trusts its experience
- (3) how much it trusts its neighbors

These parameters are usually randomly chosen, at each time step, in given intervals. Also, the neighborhood can be defined in many ways like:

- (1) Physical neighborhood, which takes distances into account. In practice, at each time step distances are recomputed, which is quite costly, but some techniques need this information.
- (2) Social neighborhood, which just takes relationships into account. In practice, for each particle, neighborhood is defined as a list of particles and does not change. Note that, when the process converges, a social neighborhood becomes a physical one.

2.3.2 Particle Swarm Optimization

The PSO is a optimization algorithm in which a given problem is solved by individual particles working together. The best thing about PSO algorithm is its simplicity. The PSO can be implemented in very small code using some of the mathematical operators, making it best for limited-memory and environment where there is computation constraint.

A basic variant of the PSO algorithm works by having a population (called a swarm) of candidate solutions (known as particles). These particles are moved around in the search-space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm's best known position. When improved positions are being discovered these will then come to guide the movements of the swarm [20]. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered. Formally, let $f: R^n \rightarrow R$ be the cost function which must be minimized. The function takes a candidate solution as argument in the form of a vector of real numbers and produces a real number as output which indicates the objective function value of the given candidate solution. The gradient of f is not known. The goal is to find a solution a for which $f(a) \leq f(b)$ for all b in the search-space, which would mean a is the global minimum [20]. Maximization can be performed by considering the function $h = -f$ instead. PSO was presented under the inspiration of bird flock immigration during the course of finding food and then be used in the optimization problems. In PSO, each optimization problem solution is taken as a bird in the searching space and it is called “particle”. Every particle has a fitness value which is determined by target functions and it has also a velocity which determines its destination and distance. All particles search in the solution space for their best positions and the

positions of the best particles in the swarm. PSO is initially a group of random particles (random solutions), and then the optimum solutions are found by repeated searching. In every iteration, a particle will follow two bests to renew itself: the best position found for a particle called pBest, the best position found for the whole swarm called gBest.

2.3.3 Basic PSO Algorithm

The basic flow for the PSO algorithm:

- (1) We begin by initializing random particle(s) (also called it population) within N-dimensional space.
- (2) As we have randomly initialized the population, so each particle will have a random velocity and thus location for each population.
- (3) Then fitness of each particle is evaluated according to the given problem. If fitness is found to be better than the particle best fitness (personal best), we will save this location for that particle as pBest (personal best). If the particle's fitness is better than the global best fitness (global best i.e. best among all the particles), we save location of this particle as gBest (global best).
- (4) Finally, we update the particle's position and velocity and look at the next particle in the population.
- (5) If the global best fitness value meets the exit criteria (satisfactory criteria- as per the problem), we end the loop and provide the location as the solution to the given problem.

3.1 Travelling Salesman Problem Using PSO

This is basically to solve Travelling Salesman Problem in which we need to find the shortest route among all cities without visiting any city twice. Let take an example of 8 cities named 0, 1, 2, 3, 4, 5, 6 and 7. So, there are 8^8 (or, 16,777,216) possible combinations, but this PSO algorithm can find the solution computing less than 8^3 solutions.

The locations of the cities we have taken in our example are (30, 5), (40, 10), (29, 25), (40, 20), (19, 25), (20, 5), (9, 19), (9, 9). This is more like a circle: the solution can be easily checked by seeing a graph and compared the algorithm's solutions with. By performing some calculations, we have found the solution to be about 86.6299, which is the target value of our algorithm.

To simplify this example, starting or ending point doesn't matter, moreover the direction of the tour travels also doesn't matter. For example, a solution that looks like 23456701 is valid because it travels forward from 2 and around to 1. The solution 65432107 is equally valid because it just goes backward from 6 to 7.

The most important part of the PSO algorithm is the calculation of the velocity vector as it is this velocity vector which updates the particle position. So, we first added the global worst to the global variables. The velocity score is then calculated using this global worst.

In this example, only three variables can be experimented with:

- (1) COUNT_OF_PARTICLE - number of particles in the test.
- (2) MAX_VELOCITY - maximum change allowed for velocity.
- (3) EPOCHS_MAX – maximum number of iterations.

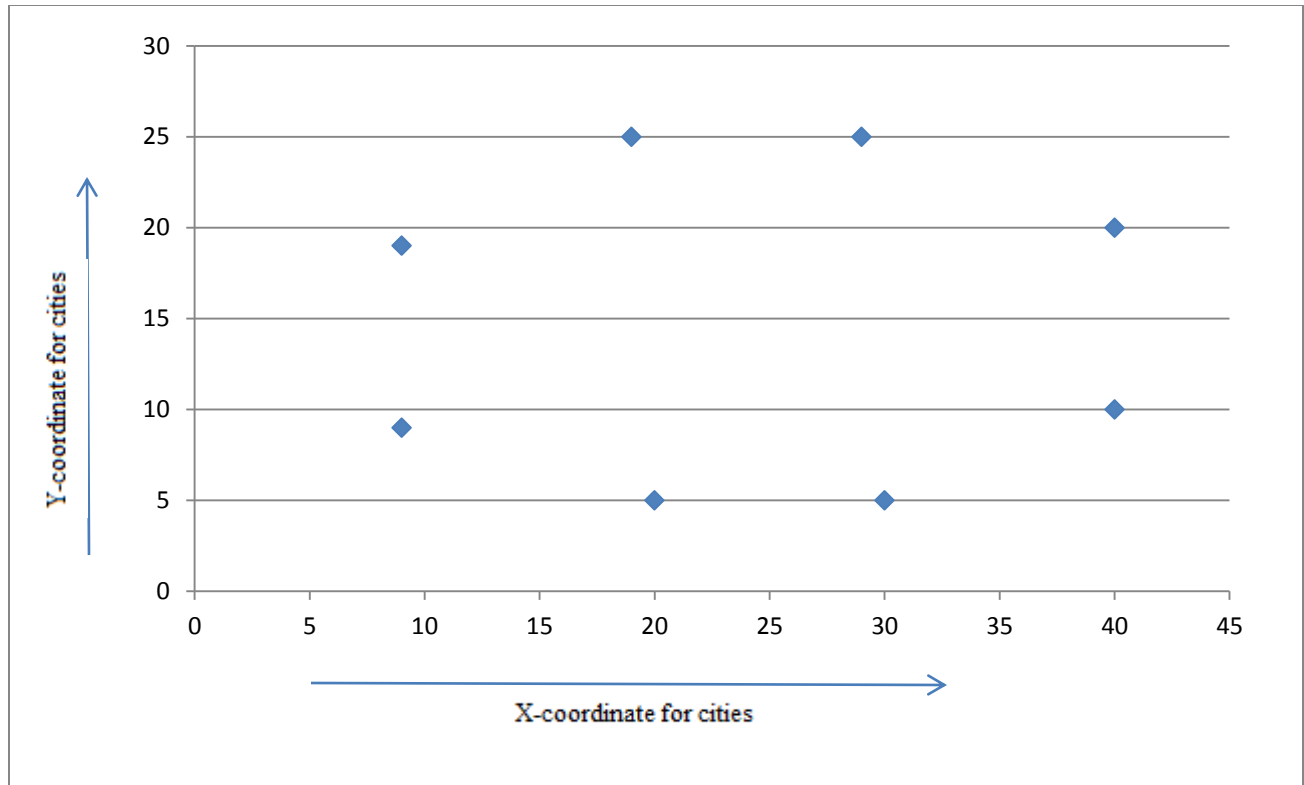


Figure 3.1: Coordinates of Cities Location

3.1.1 How it works?

Every particle has some position (x, y) and a velocity (v_x, v_y) with which it is moving through. Every particle has some tendency to move in direction same as before.

Every particle possesses some acceleration due to change in their velocity which depends upon two things:

- (1) Each particle will try to move towards the best location that it has found personally (also known as personal best-pBest).
- (2) Each particle will try to move toward the best location that any particle has found (also known as global best-gBest).

The strength with which the particles are attracted in each of these directions depends upon the parameters like pBest and gBest. PSO is used to either maximize or minimize the fitness function

by updating the position in every loop. All paths from source to destination are considered as particles.

3.1.2 Procedure

For solving Travelling Salesman Problem using Particle Swarm Optimization ,we first choose some random paths(particles). In the example we are going to discuss we have taken 4 randomly generated paths(particle).We choose these paths by randomizing the initial path value((0,1,2,3,4,5,6,7) for all the four particles).The only thing need to keep in mind while choosing these paths is that the path should be such that it should connect all points(cities) and not repeating any point(city) twice.

So

Initialize:

(1) Path 1→0 1 2 3 4 5 6 7

(2) Path 2→0 1 2 3 4 5 6 7

(3) Path 3→0 1 2 3 4 5 6 7

(4) Path 4→0 1 2 3 4 5 6 7

Now randomly rearrange each path(particle) 10 times. For rearranging these particles we select two cities for any path keeping in mind the two cities are different and swap them. We do this 10 times to increase the randomness. After randomly rearranging we get the following paths.

(1) Path 1→2 4 3 6 5 7 0 1

(2) Path 2→1 3 5 2 6 0 7 4

(3) Path 3→1 3 5 2 6 0 7 4

(4) Path 4→6 2 5 3 7 4 1 0

We calculate the distances of all these four paths(particles).

The distance comes out to be:

(1) Path 1→116.18432

(2) Path 2→180.60110

(3) Path 3→182.07424

(4) Path 4→185.11207

Now we will iterate until the maximum number of iterations has been reached or the target has been found. The maximum iteration count we have taken in this example is 10000. The target value taken by us is 86.63. If the result distance of any path comes out to be less than or equal to 86.63 we stop the iteration.

After every iteration the path value of every path changes due to change in velocity introduced. So we keep in account the personal best of each path (particle) .The personal best of each particle will be the best value (the minimum distance) the particle has encountered after n iteration.

We also need to keep track of the global best of all the particle. Global best means the best value (minimum distance) achieved so far.

The next step we are going to do is to bubble sort the paths (particles) according to their personal best scores i.e from best to worst.

We now update the particle by changing the path (particle) value.We do this with the help a very important factor, velocity. We calculate the velocity by using the formulae

Value=(MAX_VELOCITY*(personal best of path(particle))/worst result;

Here MAX_VELOCITY is the maximum velocity allowed. We have taken MAX_VELOCITY as 4.

Here worst result is the worst pBest among all paths.

This value is the indication of the no of changes to be allowed to all of the four paths so as to update them.

The values comes out to be:

(1) Path 1→2

(2) Path 2→2

(3) Path 3→3

(4) Path 4→3

Now we will again rearrange the four path according to the no of changes allowed.

After rearranging the paths comes out to be

- (1) Path 1 → 2 4 3 6 5 7 0 1 → 116.18436
- (2) Path 2 → 1 5 7 4 0 6 3 2 → 162.00802
- (3) Path 3 → 5 4 1 3 2 0 7 6 → 117.88545
- (4) Path 4 → 3 6 2 4 7 5 0 1 → 172.94305

Now these are the results after first iteration. We check whether the target value is achieved or not. We also take in account the maximum iterations.

The whole process is repeated till we reach our destination of finding the shortest path connecting all cities.

So we again bubble sort the paths (particles) and update the velocity. The updated velocity comes out to be:

- (1) Path 1 → 2
- (2) Path 2 → 2
- (3) Path 3 → 2
- (4) Path 4 → 2

Updating the paths (particles), the paths (particles) comes out to be

- (1) Path 1 → 2 3 4 7 6 5 0 1 → 110.23192
- (2) Path 2 → 2 4 3 6 5 7 0 1 → 116.18436
- (3) Path 3 → 2 4 1 3 5 7 0 6 → 114.15432
- (4) Path 4 → 6 7 4 5 3 9 1 2 → 138.37124

Again for the example we have taken the target value is not reached after the 2nd iteration. So again updating the path by applying changes to path with the velocity. After iterating 800 times we get the result in the 801 iteration.

The updated velocity comes out to be:

- (1) Path 1 → 2
- (2) Path 2 → 3
- (3) Path 3 → 2
- (4) Path 4 → 2

The updated path will be:

- (1) Path 1→2 3 4 7 6 5 0 1 →110.23192
- (2) Path 2→3 4 5 7 6 2 0 1 →131.82403
- (3) Path 3→7 0 1 2 3 4 5 6 → 86.62995
- (4) Path 4→2 4 3 6 5 7 0 1 →116.18436

Here we get the personal best of the path (particle) less than the target, we will terminate and finally get the shortest route connecting all points (cities).

The shortest path for our example is

Route →7 0 1 2 3 4 5 6

The distance for the shortest path is 86.6299.

We analyze the algorithm by changing the count of the no of paths and the velocity_max i.e. the maximum allowable change.

We get the following result.

Case 1: COUNT_OF_PARTICLE = 10, MAX_VELOCITY = 4, EPOCHS_MAX = 10000.

Route→ 2 3 4 6 5 7 0 1 → 99.9458

Route→ 2 7 3 4 6 5 0 1 → 132.2345

Route→ 6 4 7 3 5 0 1 2 → 161.0345

Route→ 4 7 0 5 2 6 1 3 → 178.4356

Route→ 3 6 1 5 0 4 7 2 → 194.8090

Route→ 4 6 0 7 2 1 5 3 → 148.6579

Route →7 0 1 2 3 4 5 6 → 86.4536

Route →5 2 3 6 7 0 1 4 → 139.6543

Route→7 2 3 6 0 4 5 1 → 171.7685

Route →2 3 0 4 6 1 5 7 → 179.8768

Changes for particle 1→ 2

Changes for particle 2→ 1

Changes for particle 3→ 4

Changes for particle 4→ 3

Changes for particle 5→ 3

Changes for particle 6→ 2

Changes for particle 7→ 4

Changes for particle 8→ 1

Changes for particle 9→ 4

epoch number: 87

Target reached.

Shortest Route: 7, 0, 1, 2, 3, 4, 5, 6, Distance: 86.4536

Case 2:COUNT_OF_PARTICLE = 10, MAX_VELOCITY = 8, EPOCHS_MAX = 10000.

Route: 0, 7, 5, 6, 4, 3, 2, 1, Distance: 99.6786

Route: 1, 0, 7, 6, 5, 4, 3, 2, Distance: 86.6345

Route: 6, 4, 1, 5, 2, 3, 0, 7, Distance: 161.5426

Route: 5, 2, 3, 0, 4, 1, 6, 7, Distance: 172.4235

Route: 1, 6, 4, 0, 2, 3, 5, 7, Distance: 162.5345

Route: 0, 1, 3, 2, 7, 5, 6, 4, Distance: 136.65456

Route: 0, 3, 6, 5, 1, 2, 7, 4, Distance: 165.6578

Route: 6, 5, 4, 7, 0, 3, 1, 2, Distance: 133.6764

Route: 7, 3, 1, 2, 4, 6, 5, 0, Distance: 136.4567

Route: 6, 5, 7, 3, 1, 4, 2, 0, Distance: 155.4568

Changes for particle 1: 5

Changes for particle 2: 3

Changes for particle 3: 6

Changes for particle 4: 8

Changes for particle 5: 2

Changes for particle 6: 5

Changes for particle 7: 7

Changes for particle 8: 6

Changes for particle 9: 8

epoch number: 223

Target reached.

Shortest Route: 1, 0, 7, 6, 5, 4, 3, 2, Distance: 86.6325

Increasing the velocity has very less effect.

3.1.3 Program Module for PSO applied on TSP

Module for applying basic PSO Algorithm:

```
private static void mainalgo()
{
    Particle aParticle = null;

    int epoch = 0;

    boolean done = false;

    initialization();

    while(!done)
    {
        // two conditions which can end this loop are:

        // the number of maximum epochs or iterations allowed has been reached, or,

        // the Target is reached.

        if(epoch < MAX_EPOCHS){
for(int i = 0; i < PARTICLE_COUNT; i++)
        {
            aParticle = particles.get(i);

            System.out.print("Route: ");

            for(int j = 0; j < CountOf_City; j++)
            {

                System.out.print(aParticle.data_Item(j) + ", ");

            }
        }
    }
}
```

```

TotalDistance(i);

if(aParticle.pBest() <= TARGET){

    done = true;

}

}

Sort(); // sorting of particles according to their pBest scores, best to worst.

newvel_ocity();

update_particles();

System.out.println("epoch number: " + epoch);

epoch++;

}

else{

    done = true;

}

}

return;

}

```

Module for initializing particles with random values(choosing random path):

```

private static void initialization()

{

for(int i = 0; i < PARTICLE_COUNT; i++)

{

    Particle newParticle = new Particle();

```

```

        for(int j = 0; j < CountOf_City; j++)
        {
            newParticle.data_Item(j, j);
        }

        particles.add(newParticle);

        for(int j = 0; j < 10; j++)
        {
            randomly_Arrange(particles.indexOf(newParticle));
        }

        TotalDistance(particles.indexOf(newParticle));
    }

    return;
}

```

Module to get velocity:

```

private static void newvel_ocity()
{
    double worstResults = 0;

    double vValue = 0.0;

    // after sorting, worst element will be the last element in list.

    worstResults = particles.get(PARTICLE_COUNT - 1).pBest();

    for(int i = 0; i < PARTICLE_COUNT; i++)
    {

```

```

vValue = (V_MAX * particles.get(i).pBest()) / worstResults;

if(vValue > V_MAX){

    particles.get(i).vel_ocity(V_MAX);

}else if(vValue < 0.0){

    particles.get(i).vel_ocity(0.0);

}else{

    particles.get(i).vel_ocity(vValue);

}

}

return;

}

```

Program module to update particle (generating new path):

```

private static void update_particles()

{

    // Best value of the particle is at index 0, so start from the second best.

    for(int i = 1; i < PARTICLE_COUNT; i++)

    {

        // The particle wil need more changes, when the vel_ocity of the particle is higher.

        int changes = (int)Math.floor(Math.abs(particles.get(i).vel_ocity()));

        System.out.println("Changes for particle " + i + ": " + changes);

        for(int j = 0; j < changes; j++){

            if(new Random().nextBoolean()){

                randomly_Arrange(i);

            }

        }

    }

}

```

```

        }
    }
    // Update the pBest value.

    getTotalDistance(i);
}
return;
}

```

Program module to get the best solution:

```

private static void printoptimalSolution()
{
    if(particles.get(0).pBest() <= TARGET){
        // Print it.

        System.out.println("Target reached.");
    }else{
        System.out.println("Target not reached");
    }

    System.out.print("Shortest Route: ");
    for(int j = 0; j < CountOf_City; j++)
    {
        System.out.print(particles.get(0).data_Item(j) + ", ");
    }
    return;
}

```

Program module to get total distance of a path (particle):

```
private static void TotalDistance(final int index)

{

Particle thisParticle = null;

thisParticle = particles.get(index);

thisParticle.pBest(0.0);


for(int i = 0; i < CountOf_City; i++)

{

if(i == CountOf_City - 1){

    thisParticle.pBest(thisParticle.pBest() + getDistance(thisParticle.data_Item(CountOf_City
- 1), thisParticle.data_Item(0))); // Full trip.

}else{

    thisParticle.pBest(thisParticle.pBest() + getDistance(thisParticle.data_Item(i),
thisParticle.data_Item(i + 1)));

}

}

return;

}
```

3.2 Genetic Algorithm for Solving TSP

Genetic algorithm is a search heuristic used in the field of artificial intelligence that explains the process of natural evolution. Genetic Algorithm belongs to a class of evolutionary algorithm. GA is applied on a set of chromosomes which are encoded as a population of chromosomes, then a

fitness function is used for the evaluation of the fitness of each chromosome, after that offspring is generated through the process of selecting a set of chromosomes known as selection, then selected chromosomes undergoes crossover to create an offspring and finally mutation. After the termination of GA, an optimal solution is found. If the optimal solution is not found depending upon the threshold value, the algorithm is run again with new population. Flowchart for proposed GA is described below-



Figure 3.2: Flowchart of Genetic Algorithm

3.2.1 Implementation of Proposed Algorithm

We have taken 15 cities and the coordinates of those cities are shown in Fig 3.3:

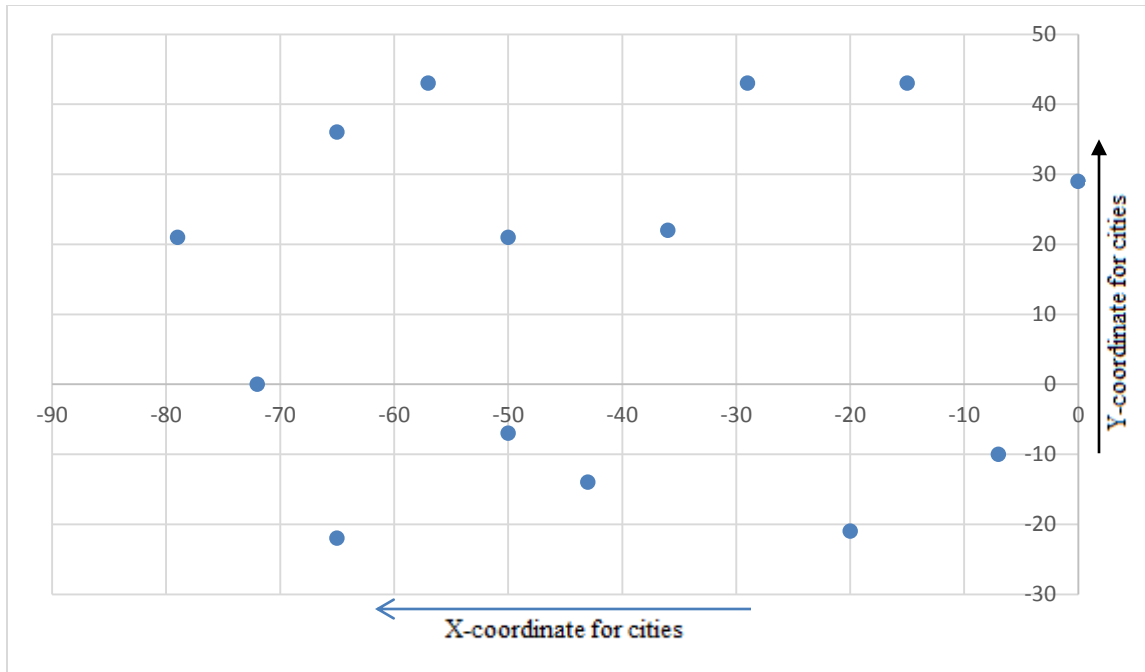


Figure 3.3: Coordinates of the 15 Cities for Solving TSP Using GA

The distance-matrix created for the distance between different cities is symmetric i.e. if we moves from city 1 to city 4 than the distance will be same if we move from city 4 to city 1. Due to this half of the entries from the matrix are omitted.

Table 3.1: Distance Matrix of 15 Cities

CITY	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	29	82	46	68	52	72	42	51	55	29	74	23	72	46
2		0	55	46	42	43	43	23	23	31	41	51	11	52	21
3			0	68	46	55	23	43	41	29	79	21	64	31	51
4				0	82	15	72	31	62	42	21	55	51	43	64
5					0	74	23	52	21	46	82	58	46	65	23
6						0	61	23	55	31	33	37	51	29	59
7							0	42	23	31	77	37	51	46	33
8								0	33	15	37	33	33	31	37
9									0	29	62	46	29	51	11
10										0	51	21	41	23	37
11											0	65	42	59	61
12												0	61	11	55
13													0	62	23
14														0	59
15															0

3.2.2 Initial Population

A unique random number generator function is used to create initial population of chromosomes. The table 3.2 shows the initial population of chromosomes. The initial population comprises of ten chromosomes. Each chromosome is made up of genes used to represent the number assigned to a city, where each chromosome denotes the sequence in which cities have to be traversed.

Table 3.2: Chromosome Path value

Chromosome1	1	4	13	3	8	2	5	15	7	10	14	12	6	9	11
Chromosome2	3	14	13	2	9	10	5	7	1	15	6	12	8	11	4
Chromosome3	1	15	3	7	14	11	9	2	13	5	12	4	8	10	6
Chromosome4	4	12	14	13	5	9	11	8	1	3	10	2	6	7	15
Chromosome5	11	2	9	5	13	14	3	12	8	1	15	6	4	10	7
Chromosome6	3	10	7	13	11	2	9	4	15	12	6	5	14	1	8
Chromosome7	11	5	2	9	15	13	7	8	4	1	3	12	6	10	14
Chromosome8	3	4	13	14	11	7	10	2	8	15	1	5	9	12	6
Chromosome9	10	11	7	8	15	1	5	9	12	4	14	6	2	13	3
Chromosome10	10	11	4	7	12	1	6	3	9	5	15	14	13	8	2

3.2.3 Fitness Value

Fitness function is used to check for the extent of goodness of a chromosome it means that it gives an idea about how good the chromosome is? The length of a chromosome is used as basic criteria for a good chromosome. Chromosomes are created by performing some calculations. Each chromosome is created and then its fitness function is calculated.

3.2.4 Selection

Chromosome with smallest fitness value is selected in the Selection method. We have used the roulette wheel selection method.

3.2.5 Roulette Wheel Selection Method

Roulette wheel selection method is used for selecting chromosomes from the given population of chromosomes because best chromosomes should be selected to generate new offspring. Parents are selected according to their fitness. The chromosomes are placed on a wheel according to their fitness value. A marble is thrown to select the chromosome and a chromosome with best fitness value will be selected more number of times.

3.2.6 Crossover

Crossover is used to generate offspring from the existing parents. Different crossover techniques are used to generate new offspring like 1-point crossover in which an offspring is generated by interchanging gene at one location within two parents. Another method used for crossover is 2-point crossover in which randomly two positions in the chromosomes are chosen and then replace the gene with each other in both chromosomes.

3.2.7 Mutation

Mutation is further applied to form a new generation. Different techniques are used for performing mutation like swap adjacent mutation in which adjacent genes a chromosome are interchanged to generate a new chromosome. We applied interchange mutation in which it randomly selects two genes in chromosome and interchange them to generate a new chromosome. Mutation is applied because there is more probability of finding better chromosome.

3.2.8 Termination and Result

The process is terminated as soon as the best tour is found. The tour after completing the number of iterations the best tour will be obtained and the process will be terminated. In figure 4.5 the tour with minimum distance is shown. The minimum distance comes out to be 291 for the problem of 15 cities.

EXPERIMENTAL RESULT AND ANALYSIS

4.1 Result Obtained after Application of PSO on TSP

The coordinates of the cities used in our problem were 0-(30,5), 1-(40,10), 2-(40,20), 3-(29,25), 4-(19,25), 5-(9,19), 6-(9,9), 7-(20,5)

After applying PSO on TSP the shortest route comes out to be 7-0-1-2-3-4-5-6. The shortest distance is 86.6299.

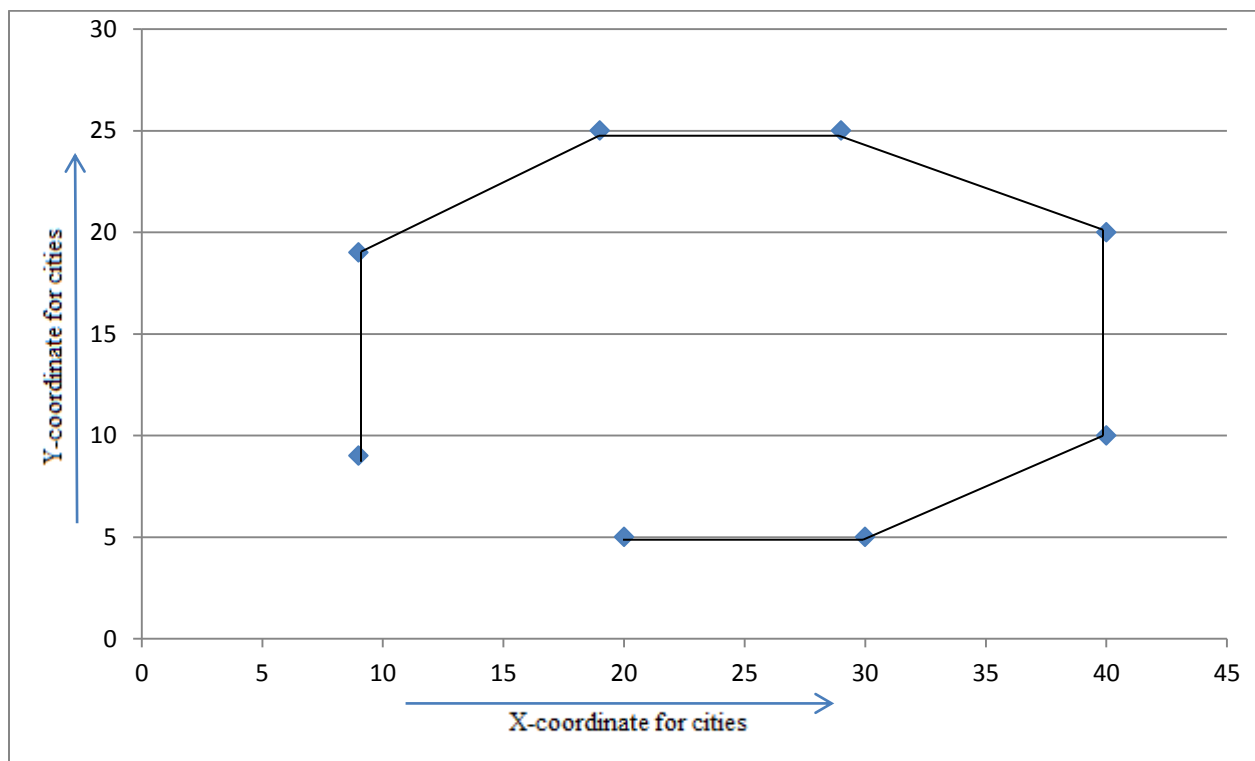
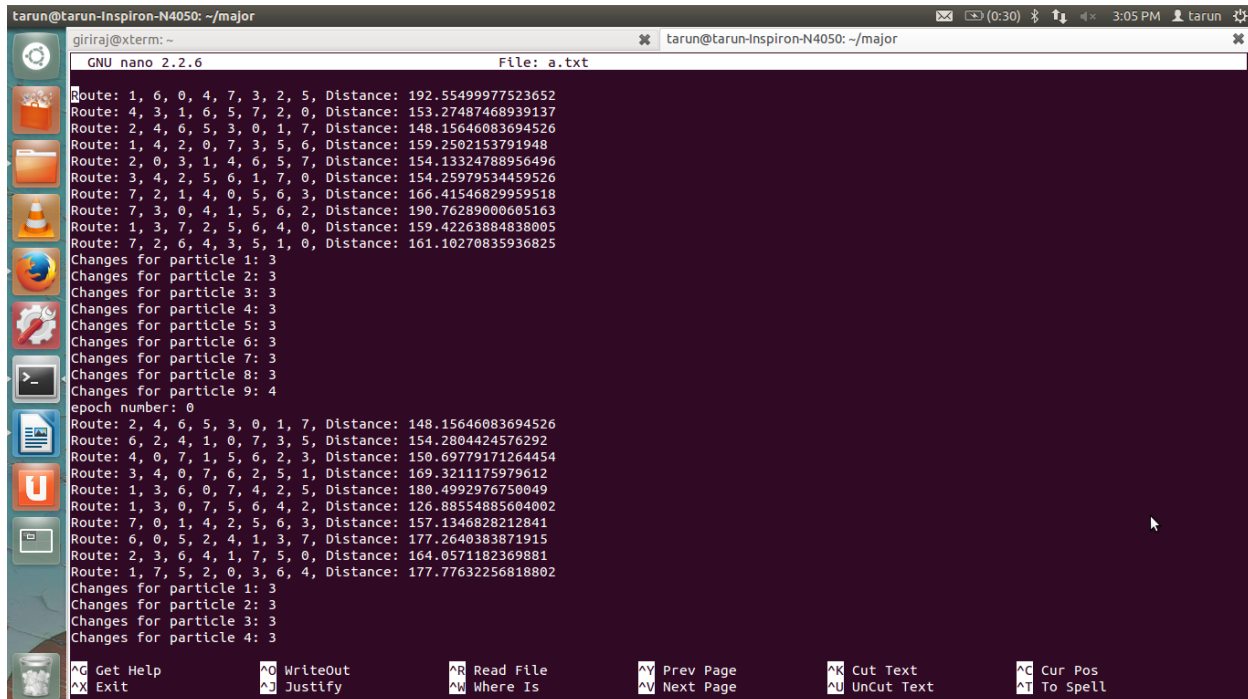


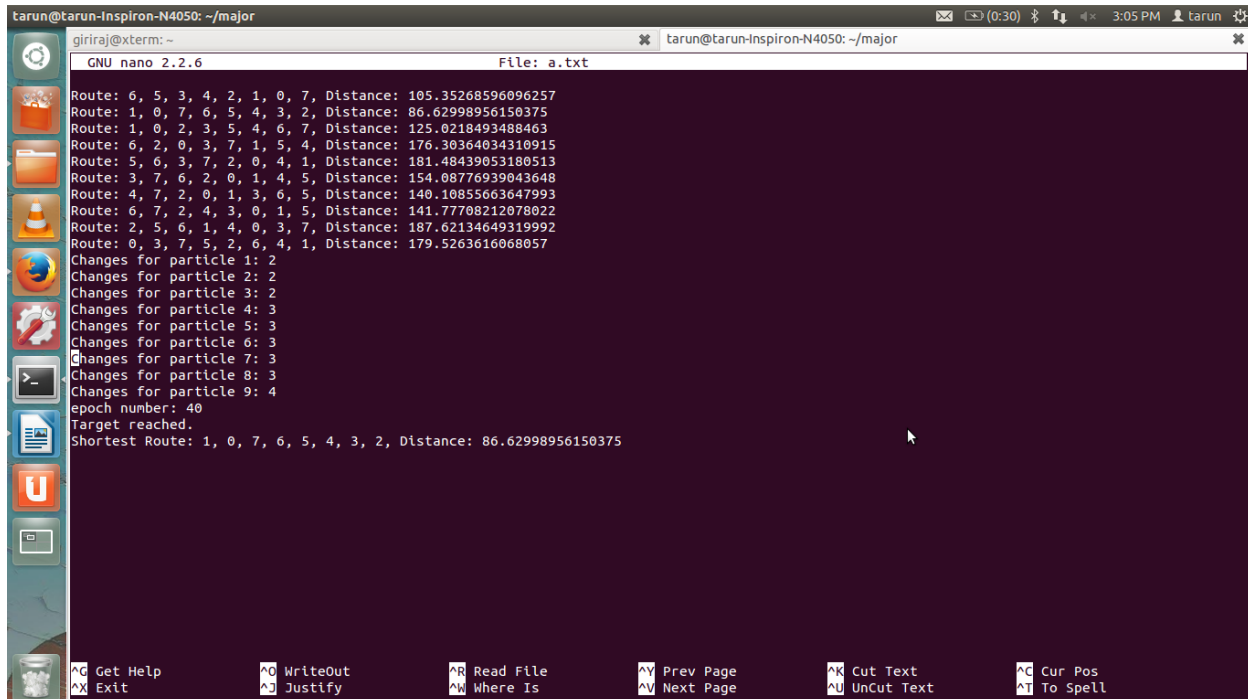
Figure 4.1: Path obtained after applying TSP using PSO

4.1.1 Output on Running PSO on Travelling Salesman Problem



```
tarun@tarun-Inspiron-N4050: ~/major
giriraj@xterm: ~
GNU nano 2.2.6 File: a.txt
Route: 1, 6, 0, 4, 7, 3, 2, 5, Distance: 192.55499977523652
Route: 4, 3, 1, 6, 5, 7, 2, 0, Distance: 153.27487468939137
Route: 2, 4, 6, 5, 3, 0, 1, 7, Distance: 148.15646083694526
Route: 1, 4, 2, 0, 7, 3, 5, 6, Distance: 159.2502153791948
Route: 2, 0, 3, 1, 4, 6, 5, 7, Distance: 154.13324788956496
Route: 3, 4, 2, 5, 6, 1, 7, 0, Distance: 154.25979534459526
Route: 7, 2, 1, 4, 0, 5, 6, 3, Distance: 166.41546829959518
Route: 7, 3, 0, 4, 1, 5, 6, 2, Distance: 190.76289000605163
Route: 1, 3, 7, 2, 5, 6, 4, 0, Distance: 159.42263884838005
Route: 7, 2, 6, 4, 3, 5, 1, 0, Distance: 161.10270835936825
Changes for particle 1: 3
Changes for particle 2: 3
Changes for particle 3: 3
Changes for particle 4: 3
Changes for particle 5: 3
Changes for particle 6: 3
Changes for particle 7: 3
Changes for particle 8: 3
Changes for particle 9: 4
epoch number: 0
Route: 2, 4, 6, 5, 3, 0, 1, 7, Distance: 148.15646083694526
Route: 6, 2, 4, 1, 0, 7, 3, 5, Distance: 154.2804424576292
Route: 4, 0, 7, 1, 5, 6, 2, 3, Distance: 150.69779171264454
Route: 3, 4, 0, 7, 6, 2, 5, 1, Distance: 169.3211175979612
Route: 1, 3, 6, 0, 7, 4, 2, 5, Distance: 180.4992976750049
Route: 1, 3, 0, 7, 5, 6, 4, 2, Distance: 126.88554885604002
Route: 7, 0, 1, 4, 2, 5, 6, 3, Distance: 157.1346828212841
Route: 6, 0, 5, 2, 4, 1, 3, 7, Distance: 177.2640383871915
Route: 2, 3, 6, 4, 1, 7, 5, 0, Distance: 164.0571182369881
Route: 1, 7, 5, 2, 0, 3, 6, 4, Distance: 177.77632256818802
Changes for particle 1: 3
Changes for particle 2: 3
Changes for particle 3: 3
Changes for particle 4: 3
^G Get Help      ^O WriteOut      ^R Read File      ^V Prev Page      ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is       ^N Next Page      ^U UnCut Text    ^T To Spell
```

Figure 4.2: Snapshot of the output of PSO for solving TSP



```
tarun@tarun-Inspiron-N4050: ~/major
giriraj@xterm: ~
GNU nano 2.2.6 File: a.txt
Route: 6, 5, 3, 4, 2, 1, 0, 7, Distance: 105.35268596096257
Route: 1, 0, 7, 6, 5, 4, 3, 2, Distance: 86.62998956150375
Route: 1, 0, 2, 3, 5, 4, 6, 7, Distance: 125.0218493488463
Route: 6, 2, 0, 3, 7, 1, 5, 4, Distance: 176.30364034310915
Route: 5, 6, 3, 7, 2, 0, 4, 1, Distance: 181.48439053180513
Route: 3, 7, 6, 2, 0, 1, 4, 5, Distance: 154.08776939043648
Route: 4, 7, 2, 0, 1, 3, 6, 5, Distance: 140.10855663647993
Route: 6, 7, 2, 4, 3, 0, 1, 5, Distance: 141.77708212078022
Route: 2, 5, 6, 1, 4, 0, 3, 7, Distance: 187.62134649319992
Route: 0, 3, 7, 5, 2, 6, 4, 1, Distance: 179.5263616068057
Changes for particle 1: 2
Changes for particle 2: 2
Changes for particle 3: 2
Changes for particle 4: 3
Changes for particle 5: 3
Changes for particle 6: 3
Changes for particle 7: 3
Changes for particle 8: 3
Changes for particle 9: 4
epoch number: 40
Target reached.
Shortest Route: 1, 0, 7, 6, 5, 4, 3, 2, Distance: 86.62998956150375
^G Get Help      ^O WriteOut      ^R Read File      ^V Prev Page      ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is       ^N Next Page      ^U UnCut Text    ^T To Spell
```

Figure 4.3: Snapshot of the output of PSO for solving TSP

4.2 Result Obtained after Application of GA on TSP

We have taken 15 cities and the coordinates of those cities are

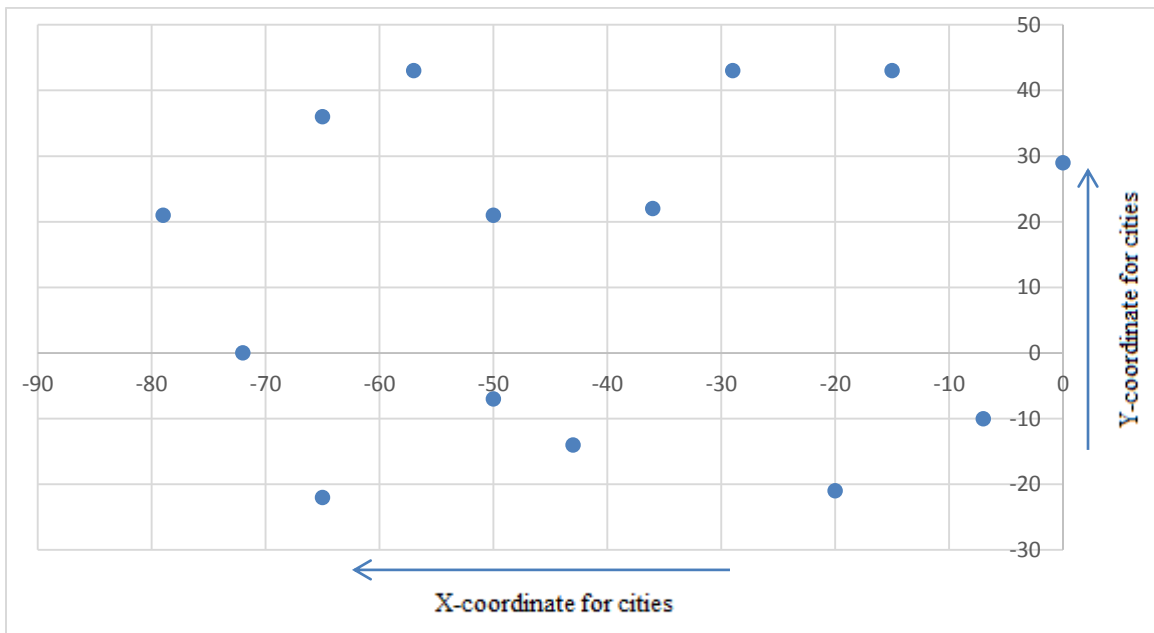


Figure 4.4: Coordinates of cities for solving TSP

After applying PSO on TSP the shortest route comes out to be

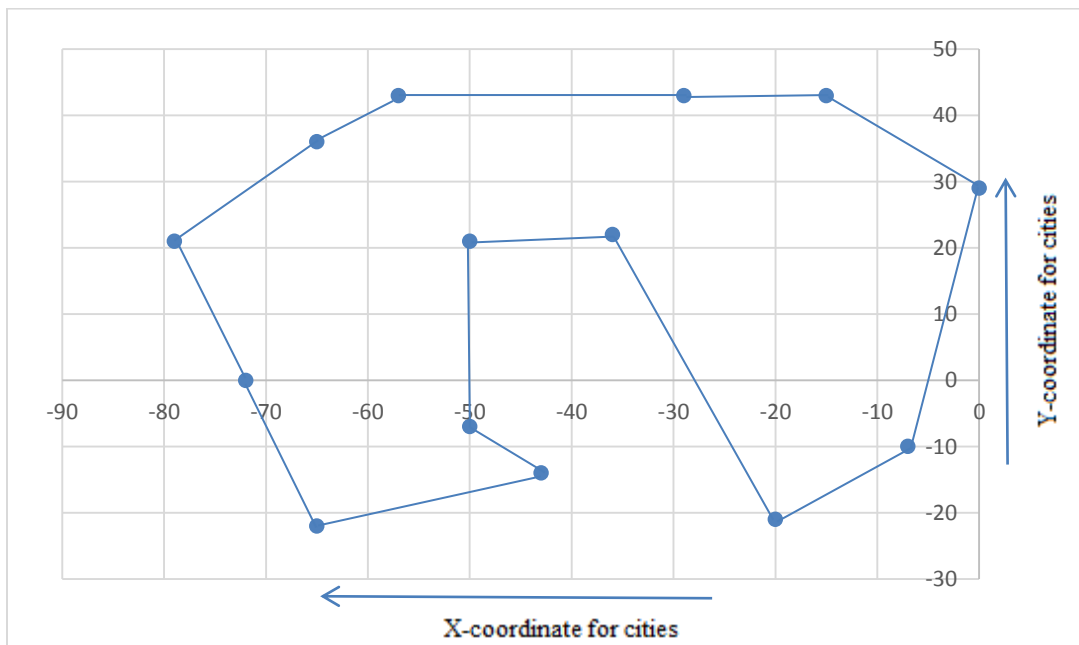
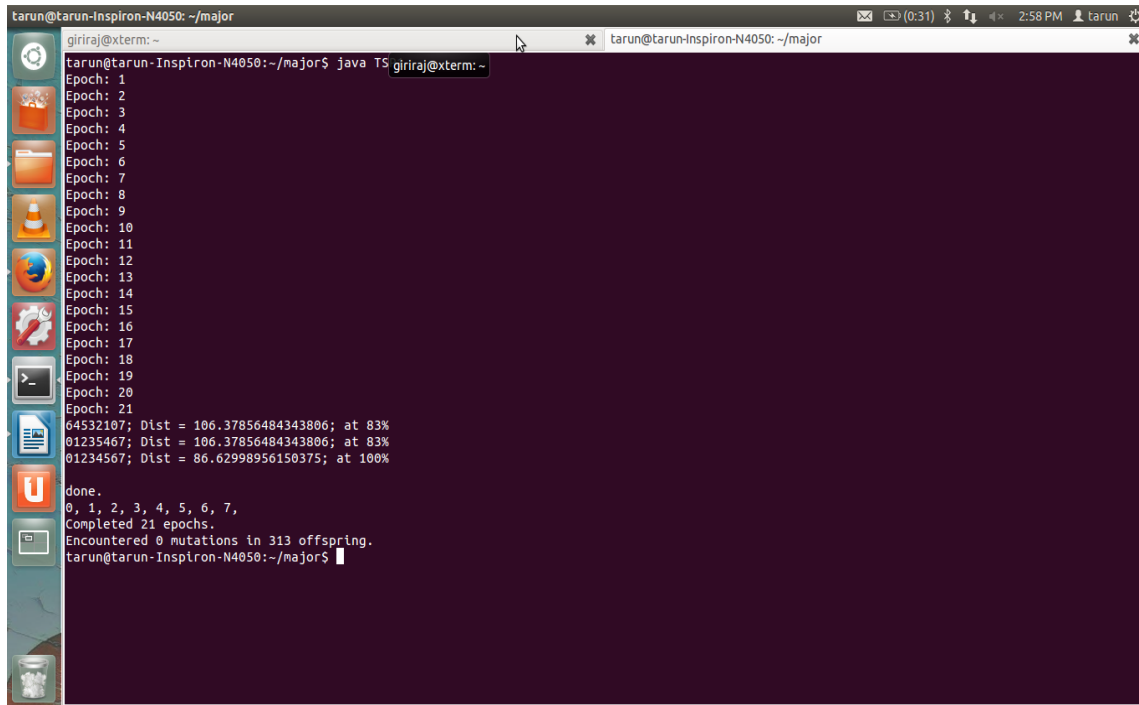


Figure 4.5: Shortest Path after applying GA on TSP

4.2.1 Output on Running Genetic Algorithm on Travelling Salesman Problem



The image shows a terminal window with a dark purple background. The window title is "tarun@tarun-Inspiron-N4050: ~/major". The prompt is "giriraj@xterm: ~". The output of the command "java TS" is as follows:

```
tarun@tarun-Inspiron-N4050:~/major$ java TS giriraj@xterm: ~
Epoch: 1
Epoch: 2
Epoch: 3
Epoch: 4
Epoch: 5
Epoch: 6
Epoch: 7
Epoch: 8
Epoch: 9
Epoch: 10
Epoch: 11
Epoch: 12
Epoch: 13
Epoch: 14
Epoch: 15
Epoch: 16
Epoch: 17
Epoch: 18
Epoch: 19
Epoch: 20
Epoch: 21
64532107; Dist = 106.37856484343806; at 83%
01235467; Dist = 106.37856484343806; at 83%
01234567; Dist = 86.62998956150375; at 100%
done.
0, 1, 2, 3, 4, 5, 6, 7,
Completed 21 epochs.
Encountered 0 mutations in 313 offspring.
tarun@tarun-Inspiron-N4050:~/major$
```

Figure 4.6: Snapshot of the output of GA for solving TSP