“USING GENETIC ALGORITHM TO SOLVE THE TRAVELING SALESMAN PROBLEM”

PROJECT REVIEW REPORT

Submitted for the course: Soft Computing (ITE1015)

By

NAME: ASTHA BARANWAL

REG. NO.: 16BIT0184

GROUP: 13

Slot G2+TG2

Name of Faculty: PROF. B. K. Tripathy

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

VIT®

UNIVERSITY

(Estd. u/s 3 of UGC Act 1956)

VELLORE ■ CHENNAI

www.vit.ac.in

CONTENTS

ACKNOWLEDGEMENT

I would like to express my gratitude towards Prof. B. K. Tripathy (Dean of the SITE School, VIT, Vellore), our Soft Computing faculty, for her kind guidance and support throughout the project.

I would like to express my special gratitude and thanks to the SITE (School of Information Technology and Engineering) School, for giving me the opportunity to make the project under the J Component section of the course, Soft Computing (ITE1015).

ASTHA BARANWAL

(16BIT0184)

ABSTRACT

The traveling salesman problem (TSP) is one of the most significant, though very hard, combinatorial optimization problem. It is the fundamental problem in the fields of computer science, engineering, operations research, discrete mathematics, graph theory, and so forth. TSP can be described as the minimization of the total distance traveled by touring all cities exactly once and return to depot city.

The problem: *"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"*

Thus, if there are 10 cities to be visited, and because once a city has been visited that city is not eligible to be travelled to again, the size of the search space stands as 10! (3628800) possible permutations. This is the reason to say TSP is NP-hard problem.

The search space for Traveling Salesman problem involves every possible permutation of routes that visit each city once. Overall, considering the total size of the search space of TSP, the genetic algorithm serves well in finding a maximal approximation of the problem.

This project will use to solve the TSP using the Genetic Algorithm. Genetic algorithms are evolutionary techniques used for optimization purposes according to survival of the fittest idea. These methods do not ensure optimal solutions; however, they give good approximation usually in time. The genetic algorithms can give near optimal solutions for NP-hard problems like TSP. The algorithm will be hard coded in Java. Since Java cannot render the solution graphically or visually, MATLAB will be used to gain the near optimal solution visually for the problem.

LITERATURE REVIEW

In the work proposed by Kylie Bryant “Genetic Algorithms and the Traveling Salesman Problem”, sGenetic algorithms are an evolutionary technique that use crossover and mutation operators to solve optimization problems using a survival of the fittest idea. [1]

In this paper proposed by Angel Goñi Moreno “Solving Travelling Salesman Problem In A Simulation Of Genetic Algorithms With Dna”, it is explained how to solve a fully connected N-City travelling salesman problem (TSP) using a genetic algorithm. A crossover operator to use in the simulation of a genetic algorithm (GA) with DNA is presented. [2]

In the work proposed by Omar M.Sallabi, “An Improved Genetic Algorithm to Solve the Traveling Salesman Problem-profound The Genetic Algorithm (GA)”, is one of the most important methods used to solve many combinatorial optimization problems. [3]

In the work proposed by Zakir H. Ahmed Genetic Algorithm for the TSP using Sequential Constructive Crossover Operator develops a new crossover operator, Sequential Constructive crossover for a genetic algorithm that generates high quality solutions to the Traveling Salesman Problem (TSP) . [4]

In the work proposed by Fozia Hanif Khan in “Solving Tsp Problem By Using Genetic Algorithm”, the main purpose of this study is to propose a new representation method of chromosomes using binary matrix and new fittest criteria to be used as method for finding the optimal solution for TSP. [5]

METHODOLOGY

SOFTWARES AND LANGUAGES

JAVA

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.

The solution to the Traveling salesman Problem (TSP) using Genetic Algorithm will be hard coded in java.

MATLAB

MATLAB is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks.

Matlab can provide a near optimal solution for the TSP problem using Genetic Algorithm approach. This optimal solution can be visualized through Matlab.

GENETIC ALGORITHM TERMS:

- **Fitness function** is a function which takes a candidate solution to the problem as input and produces as output how "fit" our how "good" the solution is with respect to the problem in consideration. **Fitness function should be sufficiently fast to compute. It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.**
- **Tournament selection** is a method of selecting an individual from a population of individuals in a genetic algorithm. Tournament selection involves running several "tournaments" among a few individuals (or "chromosomes") chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover.
- **Crossover** is a genetic operator used to combine the genetic information of two parents to generate new offspring.
- **Mutation** is used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.
- Maintaining good diversity in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as **premature convergence** and is an undesirable condition in a GA.

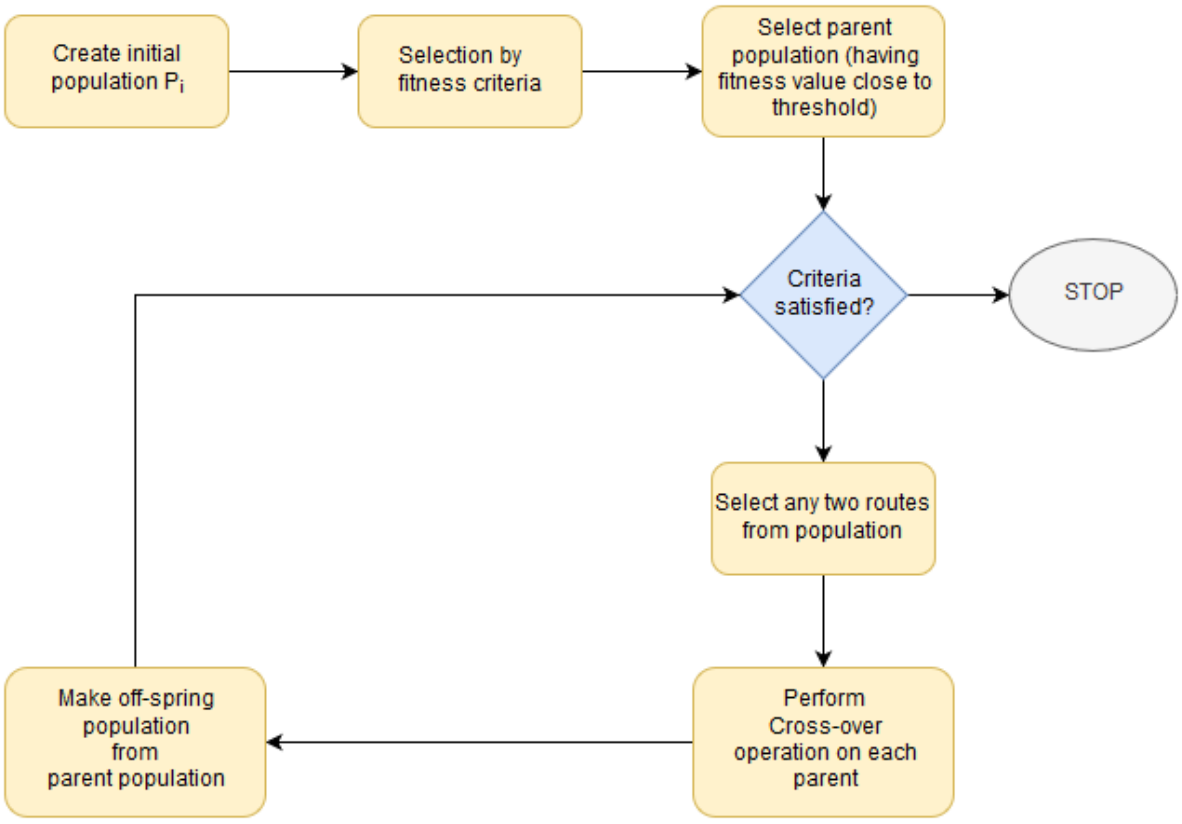
ALGORITHM STEPS

1. Randomly create the initial population of individual string of the given TSP problem and calculate distance of the path between two cities by the Haversine formula.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

2. Assign the fitness to each chromosome in the population using fitness criteria measure. Fitness measure= 1/d
3. The selection criterion depends upon the value of string if it is close to some threshold value. This finds out the good members of the population.
4. Create new off-spring population from two existing chromosomes in the parent population by applying crossover operator.
5. Mutate the resultant off-springs if required. Some of the newly formed members are altered randomly. The crossover off spring population should have fitness value higher than the parents.
6. Repeat step 3 and 4 until we get an optimal solution to the problem.

ALGORITHM FLOWCHART



THE JAVA CODE

City class

- Contains and returns information about longitude, latitude and name of the city.
- Calculates distance between two cities by the Haversine formula.

Route class

- Represents the solution which will be starting at one of the cities and passing in every city once and then returning to the originating city.
- Shuffles the cities, returns fitness value and calculates total distance travelled by taking a route.
- Fitness = (1 / Total Distance Covered by Route)*10000

Population class

- Represents a population of routes.
- Returns routes sorted in order of their fitness.

Genetic algorithm class

- Here the selection, crossover and mutation logic will take place.

Main class

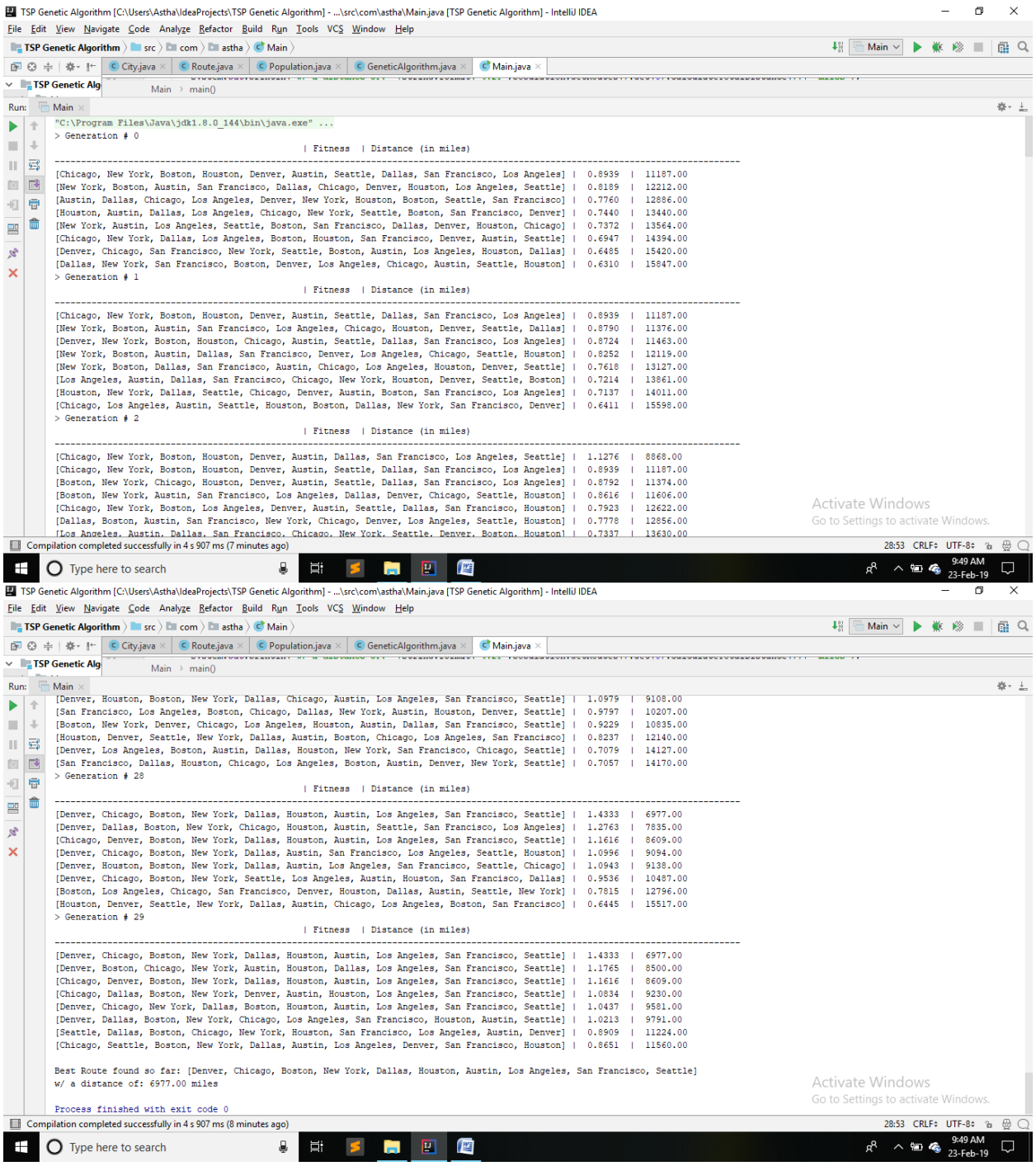
- The driver class
- Define initial route.
- Create generations.

The selection technique used is Tournament Selection. Tournament Selection is a Selection Strategy used for selecting the fittest candidates from the current generation in a Genetic Algorithm. These selected candidates are then passed on to the next generation. In a K-way tournament selection, we select k-individuals and run a tournament among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation. In this way many such tournaments take place and we have our final selection of candidates who move on to the next generation. The k in the code is 3.

The mutation rate is kept 0.25. Any two random cities (genes) are swapped two times in a route (chromosome). The population size is taken as 8.

RESULT ANALYSIS

THE JAVA CODE

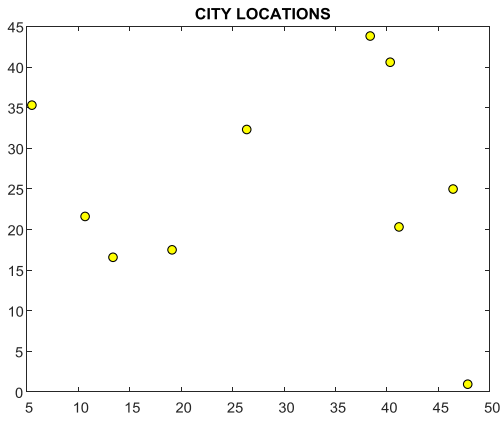


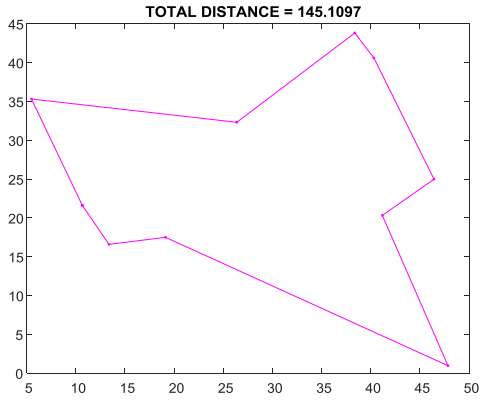
Best Route: [Denver, Chicago, Boston, New York, Dallas, Houston, Austin, Los Angeles, San Francisco, Seattle]

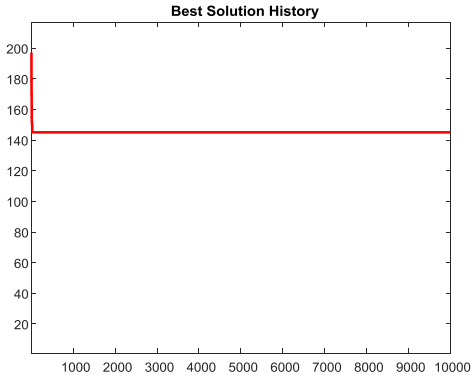
Distance of: 6977.00 miles

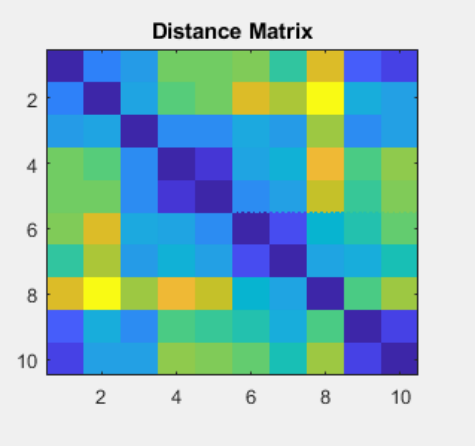
THE MATLAB CODE

Near optimal solution for 10 random cities:

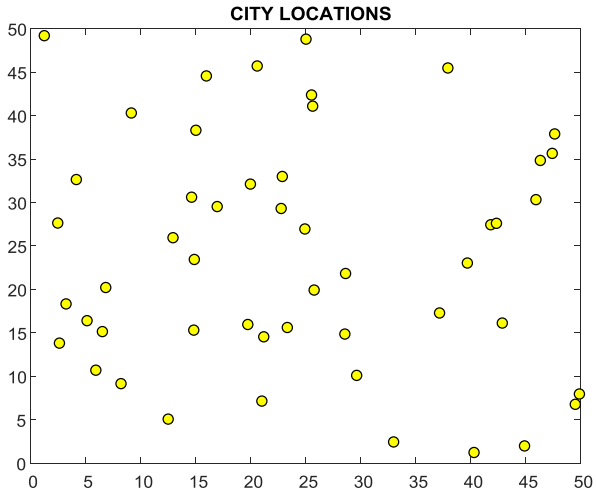


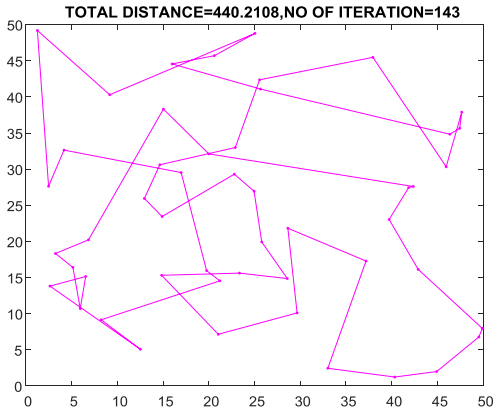


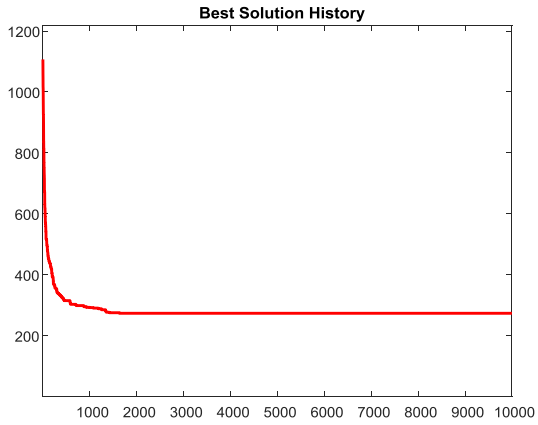


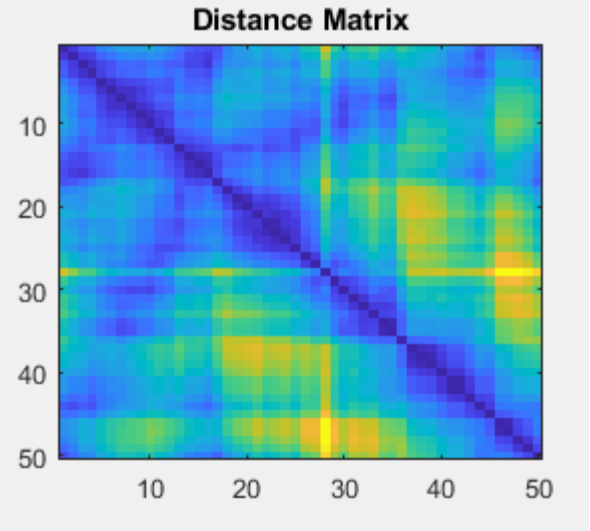


Near optimal solution for 50 random cities:

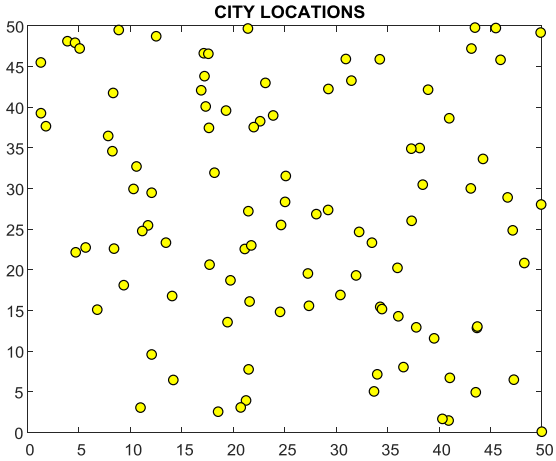


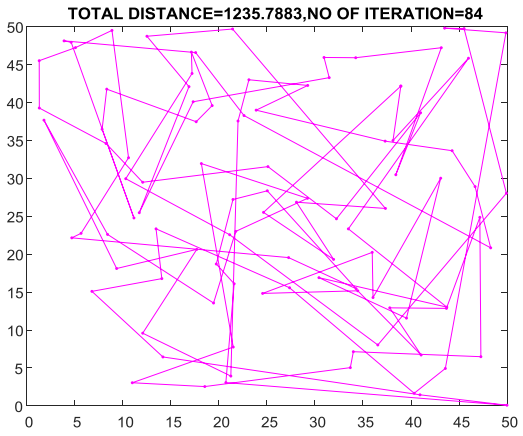


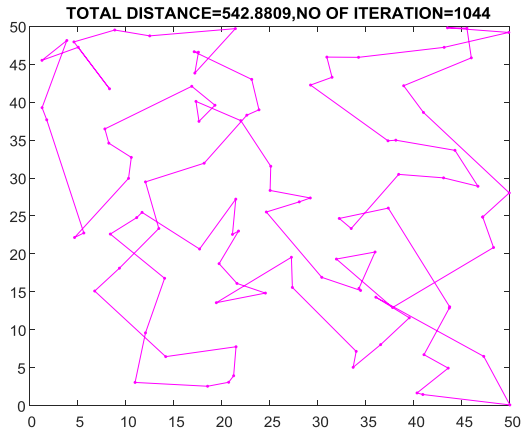


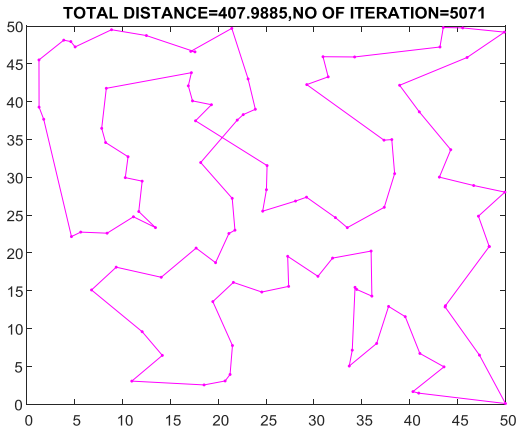


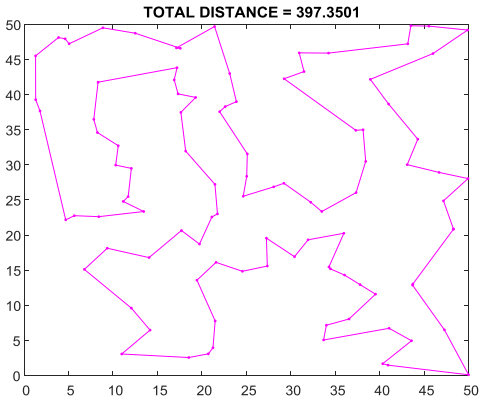
Near optimal solution for 100 random cities:

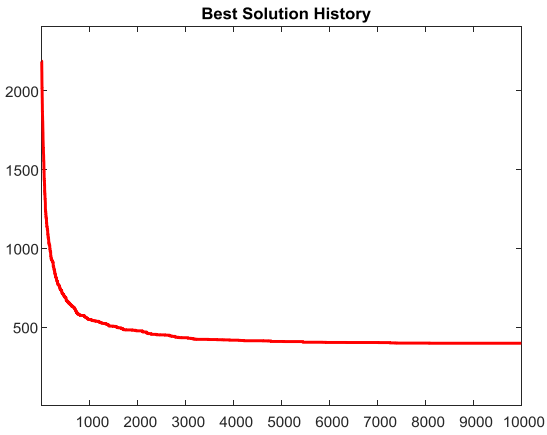


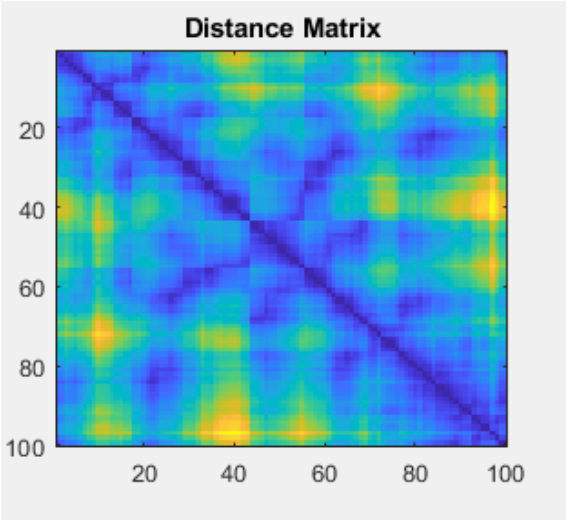

TOTAL DISTANCE=1235.7883,NO OF ITERATION=84


TOTAL DISTANCE=542.8809,NO OF ITERATION=1044

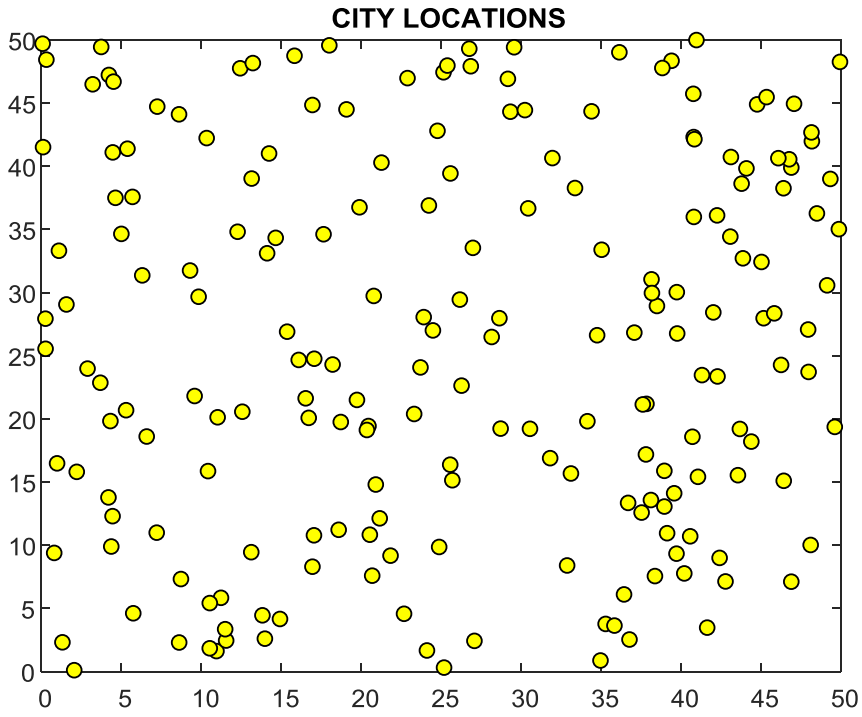

TOTAL DISTANCE=407.9885,NO OF ITERATION=5071

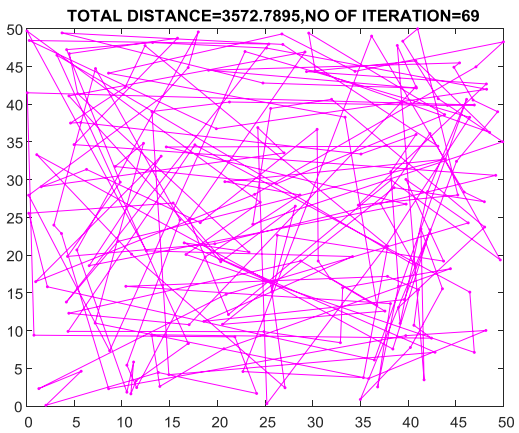

TOTAL DISTANCE = 397.3501

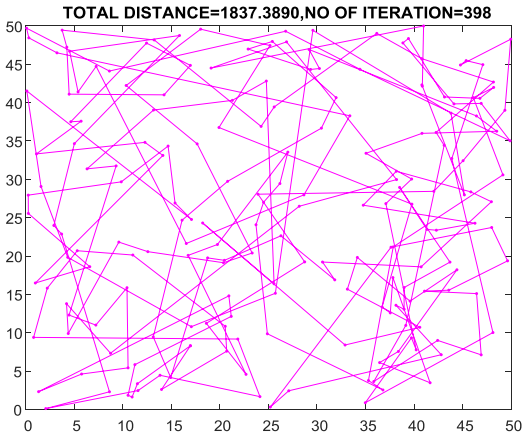

Best Solution History

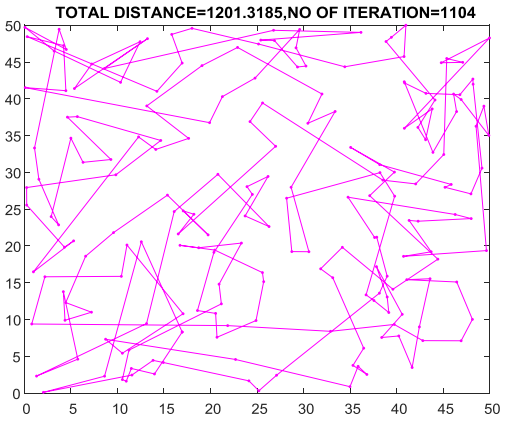

Distance Matrix

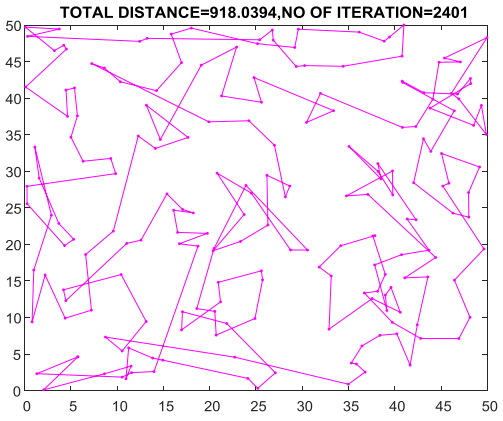
Near optimal solution for 200 random cities:

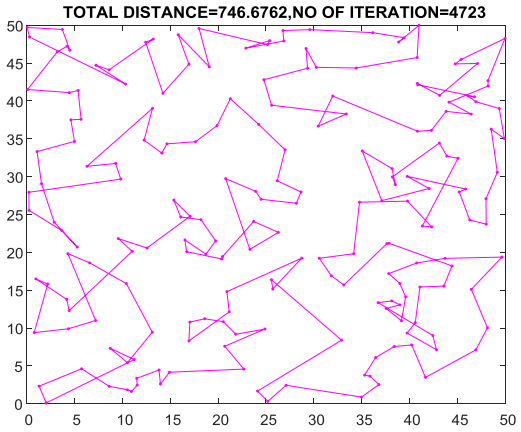


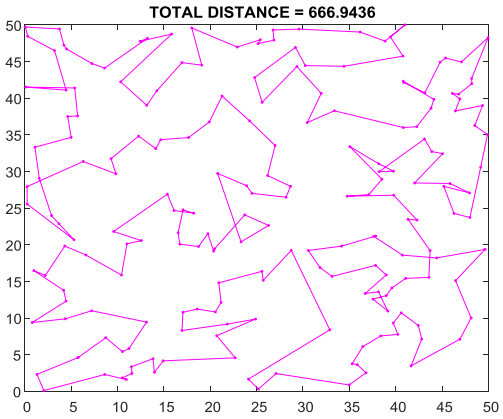


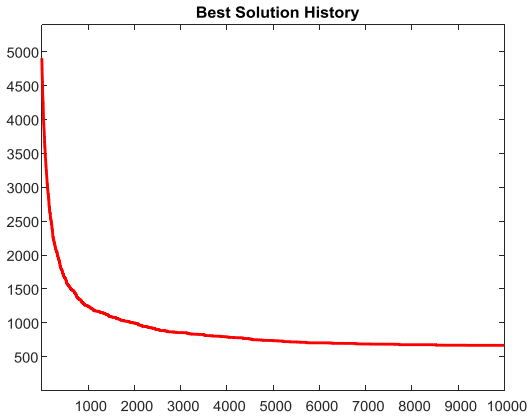


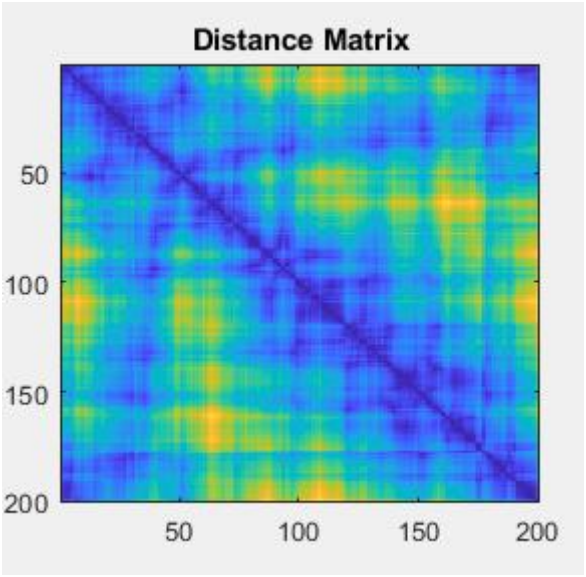

TOTAL DISTANCE=1201.3185,NO OF ITERATION=1104


TOTAL DISTANCE=918.0394,NO OF ITERATION=2401

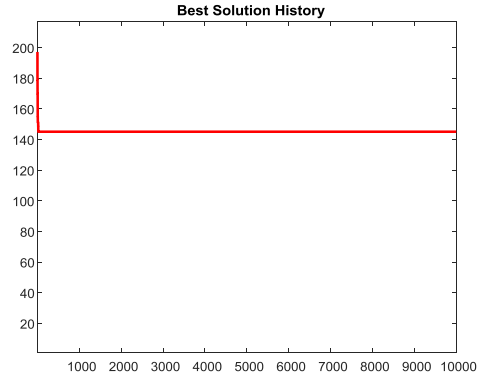

TOTAL DISTANCE=746.6762,NO OF ITERATION=4723

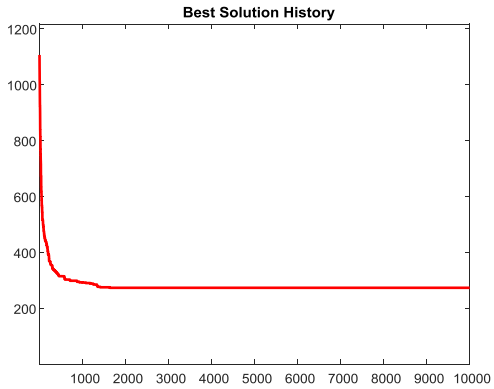

TOTAL DISTANCE = 666.9436

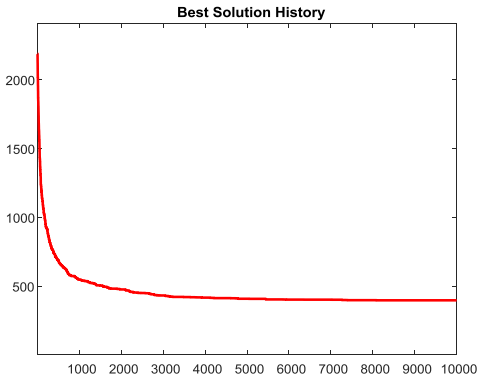

Best Solution History

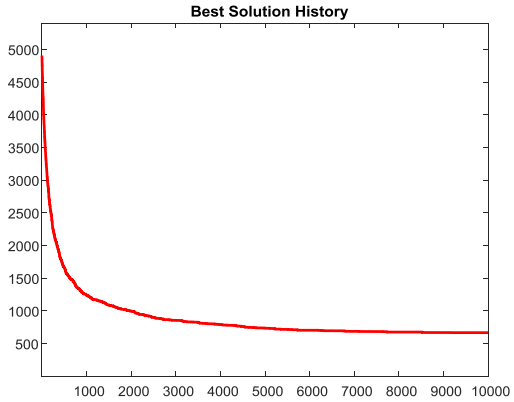

Distance Matrix

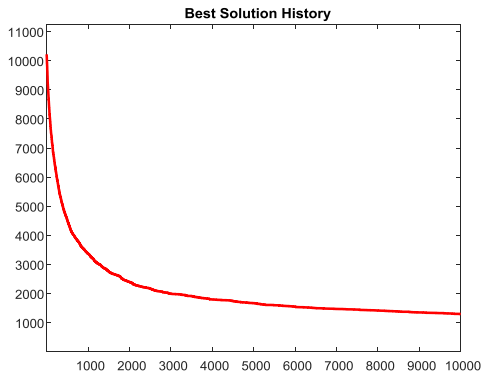
Best solution history based on fitness function for different no. of cities (10, 50, 100, 200, 400, 800):

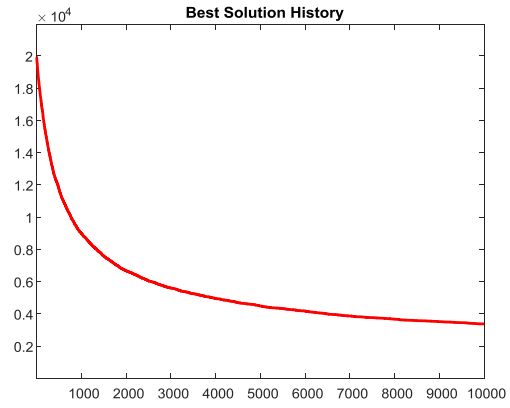










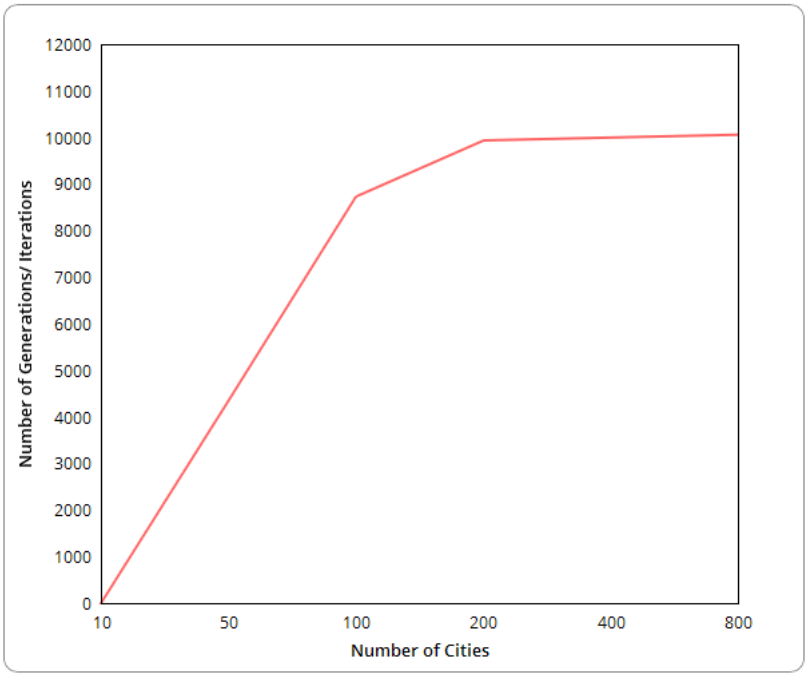


ANALYSIS OF RESULTS:

The java code was run for 10 cities. The best route found, [Denver, Chicago, Boston, New York, Dallas, Houston, Austin, Los Angeles, San Francisco, Seattle] had a distance of 6977.00 miles. Since java runs slowly and does not give pictorial representation of the routes in each generation, a MATLAB code is run.

It is clear that as the cities are increased, the number of generations/ iterations required to reach the near optimal solution also increases.

Number of cities	Number of generations/ iterations
10	7
50	4351
100	8730
200	9933
400	9995
800	10056



CONCLUSION

Genetic algorithms appear to find good solutions for the traveling salesman problem, however it depends very much on the way the problem is encoded and which crossover and mutation methods are used. It seems that the methods that use heuristic information or encode the edges of the tour (such as the matrix rep-representation and crossover) perform the best and give good indications for future work in this area.

Overall, it seems that genetic algorithms have proved suitable for solving the traveling salesman problem. As yet, genetic algorithms have not found a better solution to the traveling salesman problem than is already known, but many of the already known best solutions have been found by some genetic algorithm method also. Tournament selection was used, followed by crossover and mutation. MATLAB code gave a nearly optimal solution for different number of cities. The best solution history helped determine the trend between number of cities and the number of generations.

JAVA CODE

City.java

```
package com.astha;

public class City {
   private static final double EARTH_EQUATORIAL_RADIUS=6378.1370D;
   private static final double CONVERT_DEGREES_TO_RADIANS=Math.PI/180D;
   private static final double CONVERT_KM_TO_MILES=0.621371;
   private String name;
   private double latitude;
   private double longitude;

   public City(String name, double latitude, double longitude) {
      this.name = name;
      this.latitude = latitude * CONVERT_DEGREES_TO_RADIANS;
      this.longitude = longitude * CONVERT_DEGREES_TO_RADIANS;
   }
   public String getName(){return name;}
   public double getLatitude() {return this.latitude;}
   public double getLongitude() {return this.longitude;}

   public String toString(){ return getName();}
   public double measureDistance(City city){
      double deltaLongitude=city.getLongitude()-this.getLongitude();
      double deltaLatitude=city.getLatitude()-this.getLatitude();
      double a=Math.pow(Math.sin(deltaLatitude/2D),2D)+
            Math.cos(this.getLatitude())*Math.cos(city.getLatitude())*Math.pow(Math.sin(deltaLongitude/2D),2D);
      return CONVERT_KM_TO_MILES * EARTH_EQUATORIAL_RADIUS * 2D *
Math.atan2(Math.sqrt(a),Math.sqrt(1D-a));
   }
}
```

Route.java

```
package com.astha;

public class City {
   private static final double EARTH_EQUATORIAL_RADIUS=6378.1370D;
   private static final double CONVERT_DEGREES_TO_RADIANS=Math.PI/180D;
   private static final double CONVERT_KM_TO_MILES=0.621371;
   private String name;
   private double latitude;
   private double longitude;

   public City(String name, double latitude, double longitude) {
      this.name = name;
      this.latitude = latitude * CONVERT_DEGREES_TO_RADIANS;
      this.longitude = longitude * CONVERT_DEGREES_TO_RADIANS;
   }
   public String getName(){return name;}
   public double getLatitude() {return this.latitude;}
```

```
   public double getLongitude() {return this.longitude;}

   public String toString(){ return getName();}
   public double measureDistance(City city){
      double deltaLongitude=city.getLongitude()-this.getLongitude();
      double deltaLatitude=city.getLatitude()-this.getLatitude();
      double a=Math.pow(Math.sin(deltaLatitude/2D),2D)+
            Math.cos(this.getLatitude())*Math.cos(city.getLatitude())*Math.pow(Math.sin(deltaLongitude/2D),2D);
      return CONVERT_KM_TO_MILES * EARTH_EQUATORIAL_RADIUS * 2D *
Math.atan2(Math.sqrt(a),Math.sqrt(1D-a));
   }
}
```

Population.java

```
package com.astha;

import java.util.ArrayList;
import java.util.stream.IntStream;

public class Population {
    private ArrayList<Route> routes=new ArrayList<Route>(GeneticAlgorithm.POPULATION_SIZE);
    public Population(int populationSize, GeneticAlgorithm geneticAlgorithm){
       IntStream.range(0,populationSize).forEach(x->routes.add(new Route(geneticAlgorithm.getInitialRoute())));
   }
   public Population(int populationSize, ArrayList<City> cities){
       IntStream.range(0,populationSize).forEach(x->routes.add(new Route(cities)));
   }
   public ArrayList<Route> getRoutes(){return routes;}
   public void sortRoutesByFitness(){
       routes.sort((route1,route2)->{
          int flag=0;
          if(route1.getFitness()>route2.getFitness()) flag=-1;
          else if(route1.getFitness()<route2.getFitness()) flag=1;
          return flag;
       });
   }
}
```

GeneticAlgorithm.java

```
package com.astha;
import com.sun.org.apache.bcel.internal.generic.POP;
import org.omg.PortableServer.POA;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.stream.IntStream;

public class GeneticAlgorithm {
   public static final double MUTATION_RATE=0.25;  //probability that a chromosome gene will do random
mutation. Chromosome is a route and gene is a city in that route
   public static final int TOURNAMENT_SELECTION_SIZE=3;  //Chromosome crossover selection
```

```
   public static final int POPULATION_SIZE=8;
   public static final int NUMB_OF_ELITE_ROUTES=1; //Not subjected to crossover or mutation from one
generation to the next because of high fitness
   public static final int NUMB_OF_GENERATIONS=30; //0 to 29
   private ArrayList<City> initialRoute=null;
   public GeneticAlgorithm(ArrayList<City> initialRoute){this.initialRoute=initialRoute;}
   public ArrayList<City> getInitialRoute(){return initialRoute;}
   public Population evolve(Population population){return mutatePopulation(crossoverPopulation(population));}
   Population crossoverPopulation(Population population){
      Population crossoverPopulation=new Population(population.getRoutes().size(),this);
      IntStream.range(0,NUMB_OF_ELITE_ROUTES).forEach(x-
>crossoverPopulation.getRoutes().set(x,population.getRoutes().get(x)));
      IntStream.range(NUMB_OF_ELITE_ROUTES,crossoverPopulation.getRoutes().size()).forEach(x->{
         Route route1=selectTournamentPopulation(population).getRoutes().get(0); //highest fitness route selected
         Route route2=selectTournamentPopulation(population).getRoutes().get(0); //highest fitness route selected
         crossoverPopulation.getRoutes().set(x,crossoverRoute(route1,route2));
      });
      return crossoverPopulation;
   }
   Population mutatePopulation(Population population){
      population.getRoutes().stream().filter(x-
>population.getRoutes().indexOf(x)>=NUMB_OF_ELITE_ROUTES).forEach(x->mutateRoute(x));
      return population;
   }
   //an example crossover of route1 and route2
   //  route1                  : [New York, San Fransisco, Houston, Chicago, Boston, Austin, Seattle, Denver, Dallas, Los
Angeles]
   //  route2                  : [Los Angeles, Seattle, Austin, Boston, Denver, New York, Houston, Dallas, San Fransisco,
Chicago]
   //  intermediate crossoverRoute    : [New York, San Fransisco, Houston, Chicago, Boston, null, null, null, null, null]
   //  final crossoverRoute         : [New York, San Fransisco, Houston, Chicago, Boston, Los Angeles, Seattle, Austin,
Denver, Dallas]
   Route crossoverRoute(Route route1, Route route2){
      Route crossoverRoute=new Route(this);
      Route tempRoute1=route1;
      Route tempRoute2=route2;
      if(Math.random()<0.5){
         tempRoute1=route2;
         tempRoute2=route1;
      }
      for(int x=0;x<crossoverRoute.getCities().size()/2;x++)
         crossoverRoute.getCities().set(x,tempRoute1.getCities().get(x));
      return fillNullsInCrossoverRoute(crossoverRoute,tempRoute2);
   }
   //  crossoverRoute        : [New York, San Fransisco, Houston, Chicago, Boston, null, null, null, null, null]
   //  route              : [Los Angeles, Seattle, Austin, Boston, Denver, New York, Houston, Dallas, San Fransisco,
Chicago]
   //  (final)crossoverRoute  : [New York, San Fransisco, Houston, Chicago, Boston, Los Angeles, Seattle, Austin,
Denver, Dallas]
   private Route fillNullsInCrossoverRoute(Route crossoverRoute, Route route){

      route.getCities().stream().filter(x->!crossoverRoute.getCities().contains(x)).forEach(cityX->{
         for(int y=0;y<route.getCities().size();y++){
```

```
            if(crossoverRoute.getCities().get(y)==null){
               crossoverRoute.getCities().set(y,cityX);
               break;
            }
         }
      });
      return crossoverRoute;
   }
   //an example route mutation
   //original route: [Boston, Denver, Los Angeles, Austin, New York, Seattle, Chicago, San Fransisco, Dallas,
Houston]
   //mutated route: [Boston, Denver, New York, Austin, Los Angeles, Seattle, San Fransisco, Chicago, Dallas,
Houston]
   Route mutateRoute(Route route){
      route.getCities().stream().filter(x->Math.random()<MUTATION_RATE).forEach(cityX->{
         int y=(int)(route.getCities().size()*Math.random());
         City cityY=route.getCities().get(y);
         route.getCities().set(route.getCities().indexOf(cityX),cityY);
         route.getCities().set(y,cityX);
      });
      return route;
   }
   Population selectTournamentPopulation(Population population){
      Population tournamentPopulation=new Population(TOURNAMENT_SELECTION_SIZE,this);
      IntStream.range(0,TOURNAMENT_SELECTION_SIZE).forEach(x->tournamentPopulation.getRoutes().set(
            x,population.getRoutes().get((int)(Math.random()*population.getRoutes().size()))));
      tournamentPopulation.sortRoutesByFitness();
      return tournamentPopulation;
   }
}
```

Main.java

```
package com.astha;
import java.util.ArrayList;
import java.util.Arrays;

public class Main {
   public ArrayList<City> initialRoute=new ArrayList<City>(Arrays.asList(
         new City("Boston",42.3601,-71.0589),
         new City("Houston",29.7604 ,-95.3698),
         new City("Austin",30.2672,-97.7431),
         new City("San Francisco",37.7749,-122.4194),
         new City("Denver",39.7392,-104.9903),
         new City("Los Angeles",34.0522,-118.2437),
         new City("Chicago",41.8781,-87.6298),
         new City("New York",40.7128,-74.0059),
         new City("Dallas",32.7767,-96.7970),
         new City("Seattle",47.6062,-122.3321)
   ));
   public static void main(String[] args) {
      Main driver=new Main();
      Population population=new Population(GeneticAlgorithm.POPULATION_SIZE,driver.initialRoute);
```

```
        population.sortRoutesByFitness();
        GeneticAlgorithm geneticAlgorithm=new GeneticAlgorithm(driver.initialRoute);
        int generationNumber=0;
        driver.printHeading(generationNumber++);
        driver.printPopulation(population);
        while(generationNumber<GeneticAlgorithm.NUMB_OF_GENERATIONS){
            driver.printHeading(generationNumber++);
            population=geneticAlgorithm.evolve(population);
            population.sortRoutesByFitness();
            driver.printPopulation(population);
        }
        System.out.println("\nBest Route found so far: "+population.getRoutes().get(0));
        System.out.println("w/ a distance of:
"+String.format("%.2f",population.getRoutes().get(0).calculateTotalDistance())+" miles");
    }

    public void printPopulation(Population population){
        population.getRoutes().forEach(x->{
            System.out.println(Arrays.toString(x.getCities().toArray())+ " |  "+
            String.format("%.4f",x.getFitness())+ "  |  "+String.format("%.2f",x.calculateTotalDistance()));
        });
    }
    public void printHeading(int generationNumber){
        System.out.println("> Generation # "+generationNumber);
        String headingColumn1="Route";
        String remainingHeadingColumns="Fitness  | Distance (in miles)";
        int cityNamesLength=0;
        for(int x=0;x<initialRoute.size();x++)
            cityNamesLength+=initialRoute.get(x).getName().length();
        int arrayLength=cityNamesLength+initialRoute.size()*2;
        int partialLength=(arrayLength-headingColumn1.length())/2;
        for(int x=0;x<partialLength;x++)
            System.out.print(" ");
        if((arrayLength%2)==0) System.out.print(" ");
        System.out.println(" | "+remainingHeadingColumns);
        cityNamesLength+=remainingHeadingColumns.length()+3;
        for(int x=0;x<cityNamesLength+initialRoute.size()*2;x++)
            System.out.print("-");
        System.out.println("");

    }
}
```

MATLAB CODE

```
function varargout=TSProblem(varargin)

defaultConfig.xy=50*rand(200,2);
defaultConfig.dmat=[];
defaultConfig.popSize=150;
defaultConfig.numIter=1e4;
defaultConfig.showProg=true;
defaultConfig.showResult=true;
defaultConfig.showWaitbar=true;

if ~nargin
   userConfig=struct();
elseif isstruct(varargin(1))
   userConfig=varargin(1);
else
   try
      userConfig=struct(varargin{:});
   catch
      error('Expected inputs are either a structure or
parameter/value pairs');
   end
end

configStruct=get_config(defaultConfig,userConfig);

%Extract configuration
xy=configStruct.xy;
dmat=configStruct.dmat;
popSize=configStruct.popSize;
numIter=configStruct.numIter;
showProg=configStruct.showProg;
showResult=configStruct.showResult;
showWaitbar=configStruct.showWaitbar;

if isempty(dmat)
   nPoints=size(xy,1);
   a=meshgrid(1:nPoints);
   dmat=reshape(sqrt(sum((xy(a,:)-
xy(a',:)).^2,2)),nPoints,nPoints);
end

%Verify inputs
[N,dims]=size(xy);
[nr,nc]=size(dmat);
if N~=nr || N~=nc
   error('Invalid XY or DMAT inputs!');
end
n=N;

popSize=4*ceil(popSize/4);
numIter=max(1,round(real(numIter(1))));
showProg=logical(showProg(1));
showResult=logical(showResult(1));
showWaitbar=logical(showWaitbar(1));

%Initialize the population
pop=zeros(popSize,n);
pop(1,:)=(1:n);
for k=2:popSize
   pop(k,:)=randperm(n);
end

%Run the GA
globalMin=Inf;
totalDist=zeros(1,popSize);
distHistory=zeros(1,numIter);
tmpPop=zeros(4,n);
newPop=zeros(popSize,n);
if showProg
   figure('Name','TSP_GA | Current Best
Solution','Numbertitle','off');
   hAx=gca;
end
if showWaitbar
   hWait=waitbar(0,'Searching for near-optimal
solution ...');
end
for iter=1:numIter
   %Evaluate each population member (Calculate
total distance)
   for p=1:popSize
      d=dmat(pop(p,n),pop(p,1));%Closed path
      for k=2:n
         d=d+dmat(pop(p,k-1),pop(p,k));
      end
      totalDist(p)=d;
   end

   %Find the best route in the population
   [minDist,index]=min(totalDist);
   distHistory(iter)=minDist;
   if minDist<globalMin
      globalMin=minDist;
      optRoute=pop(index,:);
      if showProg
         %Plot the best route
         rte=optRoute([1:n 1]);
         if
dims>2,plot3(hAx,xy(rte,1),xy(rte,3),'MarkerEdgeCol
or','r');
```

```
        else plot(hAx,xy(rte,1),xy(rte,2),'m.-');end
        title(hAx,sprintf('TOTAL DISTANCE=%1.4f,NO
OF ITERATION=%d',minDist,iter));
        drawnow;
      end
    end

    %Genetic Algorithm Operators
    randomOrder=randperm(popSize);
    for p=4:4:popSize
      rtes=pop(randomOrder(p-3:p),:);
      dists=totalDist(randomOrder(p-3:p));
      [ignore,idx]=min(dists);
      bestOf4Route=rtes(idx,:);
      routeInsertionPoints=sort(ceil(n*rand(1,2)));
      I=routeInsertionPoints(1);
      J=routeInsertionPoints(2);
      for k=1:4 %Mutate the best to get 3 new routes
        tmpPop(k,:)=bestOf4Route;
        switch k
          case 2 %Flip
            tmpPop(k,I:J)=tmpPop(k,J:-1:I);
          case 3 %Swap
            tmpPop(k,[I,J])=tmpPop(k,[J,I]);
          case 4 %Slide
            tmpPop(k,I:J)=tmpPop(k,[I+1:J I]);
          otherwise
        end
      end
      newPop(p-3:p,:)=tmpPop;
    end
    pop=newPop;

    %Update the waitbar
    if showWaitbar && ~mod(iter,ceil(numIter/325))
      waitbar(iter/numIter,hWait);
    end
  end

  if showWaitbar
    close(hWait)
  end

  if showResult
    %Plot the GA results
    figure('Name','TSP_GA |
Results','Numbertitle','off');
    % subplot(2,1,1);
    pclr=~get(0,'DefaultAxesColor');
    if dims>2,plot3(xy(:,1),xy(:,2),xy(:,3),'.','Color',[1 1
0],pclr);
    else
plot(xy(:,1),xy(:,2),'o','MarkerFaceColor','y','Color',pcl
r); end
    title('CITY LOCATIONS');
    pause(0.9)
    figure('Name','TSP_GA1 |
Results','Numbertitle','off');
    % subplot(2,1,1)
    rte=optRoute([1:n 1]);
    if dims>2,plot3(xy(rte,1),xy(rte,2),xy(rte,3),'m.-');
    else plot(xy(rte,1),xy(rte,2),'m.-');end
    title(sprintf('TOTAL DISTANCE = %1.4f',minDist));

    pause(0.9)
    figure('Name','TSP_GA2 |
Results','Numbertitle','off');
    % subplot(2,1,1)
    rte=optRoute([1:n 1]);
    if dims>2,plot3(xy(rte,1),xy(rte,2),xy(rte,3),'m.-');
    else plot(xy(rte,1),xy(rte,2),'m.-');end
    title(sprintf('TOTAL DISTANCE = %1.4f',minDist));

    pause(0.9)
    figure('Name','TSP_GA3 |
Results','Numbertitle','off');
    % subplot(2,1,2)
    plot(distHistory,'r','LineWidth',2);
    title('Best Solution History');
    set(gca,'XLim',[1 numIter+1],'YLim',[1 1.1*max([1
distHistory])]);

    pause(0.9)
    figure('Name','TSP_GA |
Results','Numbertitle','off');
    % subplot(2,2,3)
    pclr=~get(0,'DefaultAxesColor');
    if dims>2,plot3(xy(:,1),xy(:,2),xy(:,3),'.','Color',pclr);
    else plot(xy(:,1),xy(:,2),'.','Color',pclr);end
    title('City Locations');
    subplot(2,2,4);
    imagesc(dmat(optRoute,optRoute));
    title('Distance Matrix');
    subplot(2,2,2);
    rte=optRoute([1:n 1]);
    if dims>2,plot3(xy(rte,1),xy(rte,2),xy(rte,3),'r.-');
    else plot(xy(rte,1),xy(rte,2),'r.-');end
    title(sprintf('Total Distance = %1.4f',minDist));
    subplot(2,2,1);
    plot(distHistory,'b','LineWidth',2);
    title('Best Solution History');
    set(gca,'XLim',[0 numIter+1],'YLim',[0 1.1*max([1
distHistory])]);
  end
```

```
% return output
if nargout
   resultStruct=struct( ...
      'xy',xy, ...
      'dmat',dmat, ...
      'popSize',popSize, ...
      'numIter',numIter, ...
      'showProg',showProg, ...
      'showResult',showResult, ...
      'showWaitbar',showWaitbar, ...
      'optRoute',optRoute, ...
      'minDist',minDist);
   varargout=(resultStruct);
end
end

function config=get_config(defaultConfig,userConfig)
   config=defaultConfig;
   defaultFields=fieldnames(defaultConfig);
   userFields=fieldnames(userConfig);
   nUserFields=length(userFields);
   for i=1:nUserFields
      userField=userFields{i};
      isField=strcmpi(defaultFields,userField);
      if nnz(isField)==1
         thisField=defaultFields{isField};
         config.(thisField)=userConfig.(userField);
      end
   end
end
```

REFERENCES

[1] Kylie Bryant, Arthur Benjamin, Advisor, "Genetic Algorithms and the Travelling Salesman Problem", Department of Mathematics, December 2000.

[2] Angel Goñi Moreno, "Solving Travelling Salesman Problem In A Simulation Of Genetic Algorithms With Dna", International Journal "Information Theories & Applications" Vol.15 / 2008

[3] Omar M. Sallabi, "An Improved Genetic Algorithm to Solve the Travelling Salesman Problem", World Academy of Science, Engineering and Technology, 2009

[4] Fozia Hanif Khan, Nasiruddin Khan, Syed Inaya Tullah, And Shaikh Tajuddin Nizami, "Solving TSP problem by using Genetic Algorithm", International Journal of Basic &Applied Sciences IJBAS Vol: 9 No: 10, August 2010

[5] Zakir H. Ahmed , "Genetic Algorithm for the TSP using Sequential Constructive Crossover Operator", International Journal of Biometrics & Bioinformatics (IJBB) Volume (3): Issue (6), April 2018