**Solving Travelling Salesman Problem using the Genetic Algorithm**

**Research Background:**

Travelling salesman problem is describing as follows:

*“Given a list of cities and the distances between each pair of cities, what is the shortest and fastest possible route that visits each city and returns to the origin city?”*

Thus in this problem, the objective is to minimize the overall distance traveled by tourists. There should be 2 main points which should consider.

1. Each city needs to be visited exactly one time.
2. We must return to the starting city, so our total distance needs to be calculated accordingly.

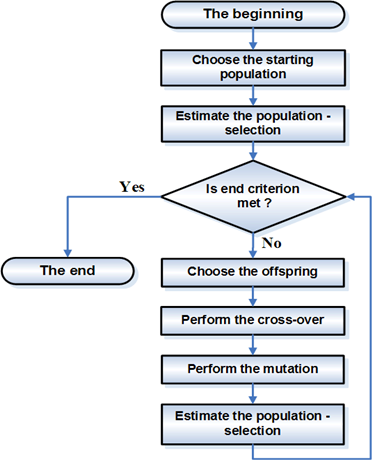
However, when only the overall distance is minimized, depending on the distribution of the cities, some travelers tend to travel much more than others. Therefore, the development of Genetic Algorithms (GAs) is implemented to reduce both the overall distance and the difference between the distance traveled by tourists. The purpose of this report is to analyze the applicability of using GAs to solve this problem considering the overall distance and also the balance of the tourist’s routes by using Matlab.

**Research Method:**

In [computer science](https://en.wikipedia.org/wiki/Computer_science) and [operations research](https://en.wikipedia.org/wiki/Operations_research), a genetic algorithm (GA) is a [metaheuristic](https://en.wikipedia.org/wiki/Metaheuristic) inspired by the process of [natural selection](https://en.wikipedia.org/wiki/Natural_selection) that belongs to the larger class of [evolutionary algorithms](https://en.wikipedia.org/wiki/Evolutionary_algorithm) (EA). It is a perfect optimization technique. In genetic algorithms we will be using [mutation](https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)), [crossover](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)) and [selection](https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)) processes and they are called as Genetic operators. Subsequent generations will evolve from following these three genetic operators.

The evolution will start from a population of fixed data, and the [fitness](https://en.wikipedia.org/wiki/Fitness_(biology)) of every individual in the population will be evaluated. Then, the new generation of candidate solutions is then used in the next iteration of the [algorithm](https://en.wikipedia.org/wiki/Algorithm). Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

This is the flowchart of the Genetic Algorithm.

**

**Choosing the starting population**

First fitness values for each chromosome will be calculated. And based on that the starting population will be selected.

**Selection**

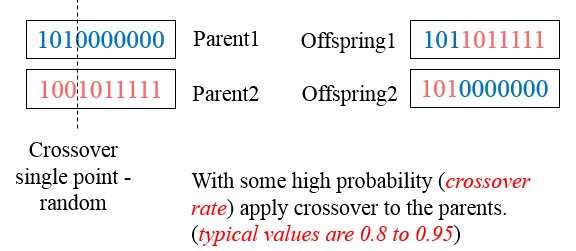
Selection retrains best performing bit strings from one generation to the next. It favors these for reproduction. Some of the methods use for selection are Roulette wheel selection, Stochastic universal sampling(SUS) and tournament selection.

In Roulette wheel selection, we will spin the wheel and whenever the wheel stops, the individual at that point will get selected. The individual that has the highest fitness value gets selected larger share of the wheel.

**Crossover – Recombination**

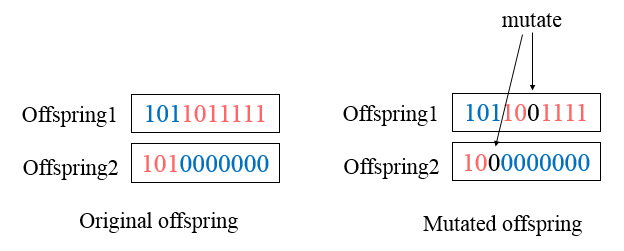
Here the selected chromosome will be taking for mating and mixes the genetic material to produce offspring.

Some of the popular crossover operations are one-point crossover, multi-point crossover, uniform crossover etc.



**Mutation**

Mutation introduce diversity within the population so that search algorithm doesn’t necessarily get stuck at local minima. Some of the mutation operations are bit flip mutation, swap mutation, scramble operation, inversion mutation etc.



This process will be repeated until the termination criteria is reached.

**Solution to the Travelling Salesman Problem:**

Source code:

clear all;

load('X.mat');

city = X;

plot(city(:,3),city(:,2),'s','MarkerSize',5,'MarkerEdgeColor','b','MarkerFaceColor',[0,0.7,0.9]),axis equal;

cityAmount = size(city,1);

PopSize = 100;

Pc = 0.90;

Pm = 0.02;

for i = 1:100

Chrom(i, :) = randperm(cityAmount);

end

for kk = 1:1000

for i = 1:100

fitness(i) = 0;

for d = 1:cityAmount

StartingCity = Chrom(i,d);

DestinationCity = Chrom(i,rem(d,cityAmount)+1);

fitness(i) = fitness(i) + sqrt((city(DestinationCity,2)-city(StartingCity,2))^2 + (city(DestinationCity,3)-city(StartingCity,3))^2);

end

RT(i) = sum(10000./fitness(1:i));

end

maxfit(kk) = max(10000./fitness);

if kk==1

QQQ=find((10000./fitness)==max(10000./fitness));

BestChrom=Chrom(QQQ(1),:);

else

if maxfit(kk)>maxfit(kk-1)

QQQ=find((10000./fitness)==max(10000./fitness));

BestChrom=Chrom(QQQ(1),:);

end

end

Chrom(1,:)=BestChrom;

if kk>15 && abs(maxfit(kk)-maxfit(kk-14))<10^(-15)

break

end

for i=1:30

ReChrom(i,:) = BestChrom;

end

for i=31:100

a=find(RT>=rand\*sum(10000./fitness));

ReChrom(i,:)=Chrom(a(1),:);

end

%Crossover

CrChrom=ReChrom;

for i=2:2:PopSize\*Pc

Crossover=randi([2 (cityAmount/2)]);

for j = 1:Crossover

index\_s = find(CrChrom(i-1,:)==ReChrom(i,j));

CrChrom(i-1,index\_s)= CrChrom(i-1,j);

CrChrom(i-1,j)= ReChrom(i,j);

index\_t = find(CrChrom(i,:)==ReChrom(i-1,j));

CrChrom(i,index\_t)= CrChrom(i,j);

CrChrom(i,j)= ReChrom(i-1,j);

end

end

%Mutation

ChromTemp = CrChrom;

for i=1:PopSize\*Pm\*cityAmount

Mutation=randperm(cityAmount,2);

if (Mutation(1)>Mutation(2))

temp\_mutation = Mutation(2);

Mutation(2) = Mutation(1);

Mutation(1) = temp\_mutation;

end

CrChrom(i,Mutation(1):Mutation(2)) = fliplr(ChromTemp(i,Mutation(1):Mutation(2)));

end

Chrom=CrChrom;

end

Best\_Distance = 0;

for i = 1:cityAmount

StartingCity = BestChrom(i);

DestinationCity = BestChrom(rem(i, cityAmount) + 1);

Best\_Distance = Best\_Distance + sqrt((city(DestinationCity,2)-city(StartingCity,2))^2 + (city(DestinationCity,3)-city(StartingCity,3))^2);

ans(i,:) = city(BestChrom(i),:);

end

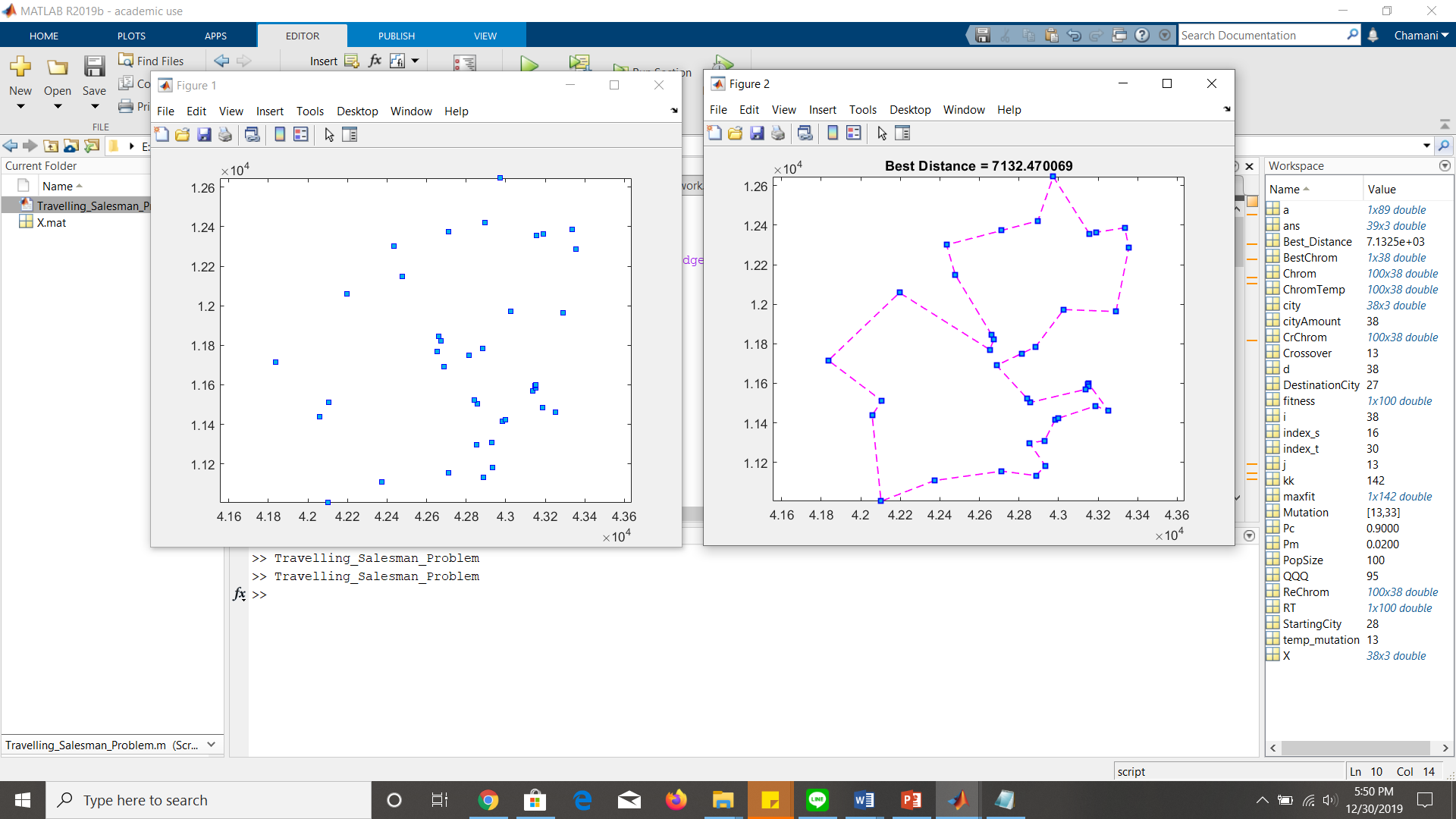
ans(i+1,:) = ans(1,:);

figure

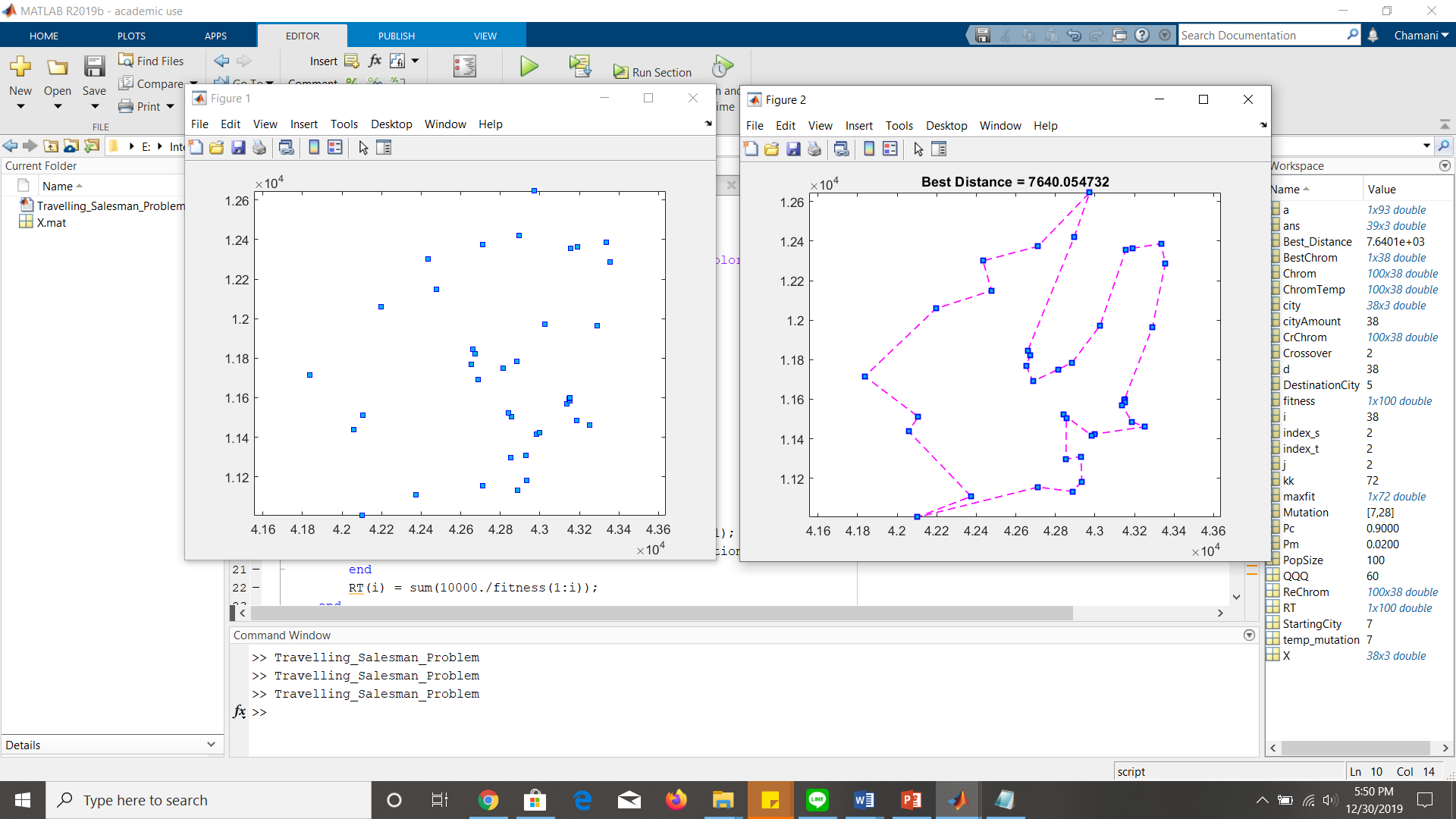
plot(ans(:,3), ans(:,2), '--sm', 'LineWidth', 1, 'MarkerSize',5,'MarkerEdgeColor','b','MarkerFaceColor',[0,0.7,0.9]),axis equal;

title(sprintf('Best Distance = %f', Best\_Distance));

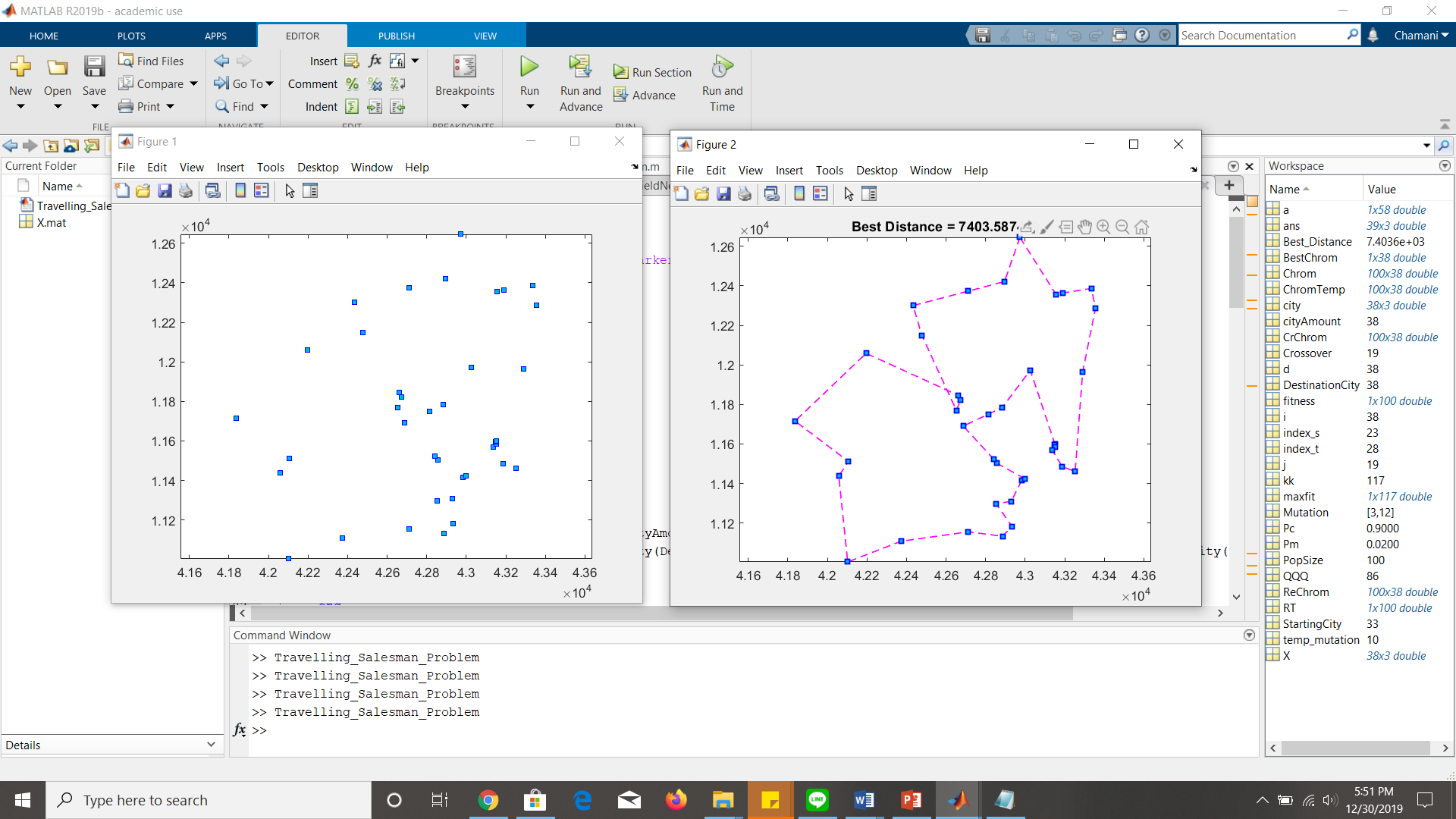
**1st experiment:**



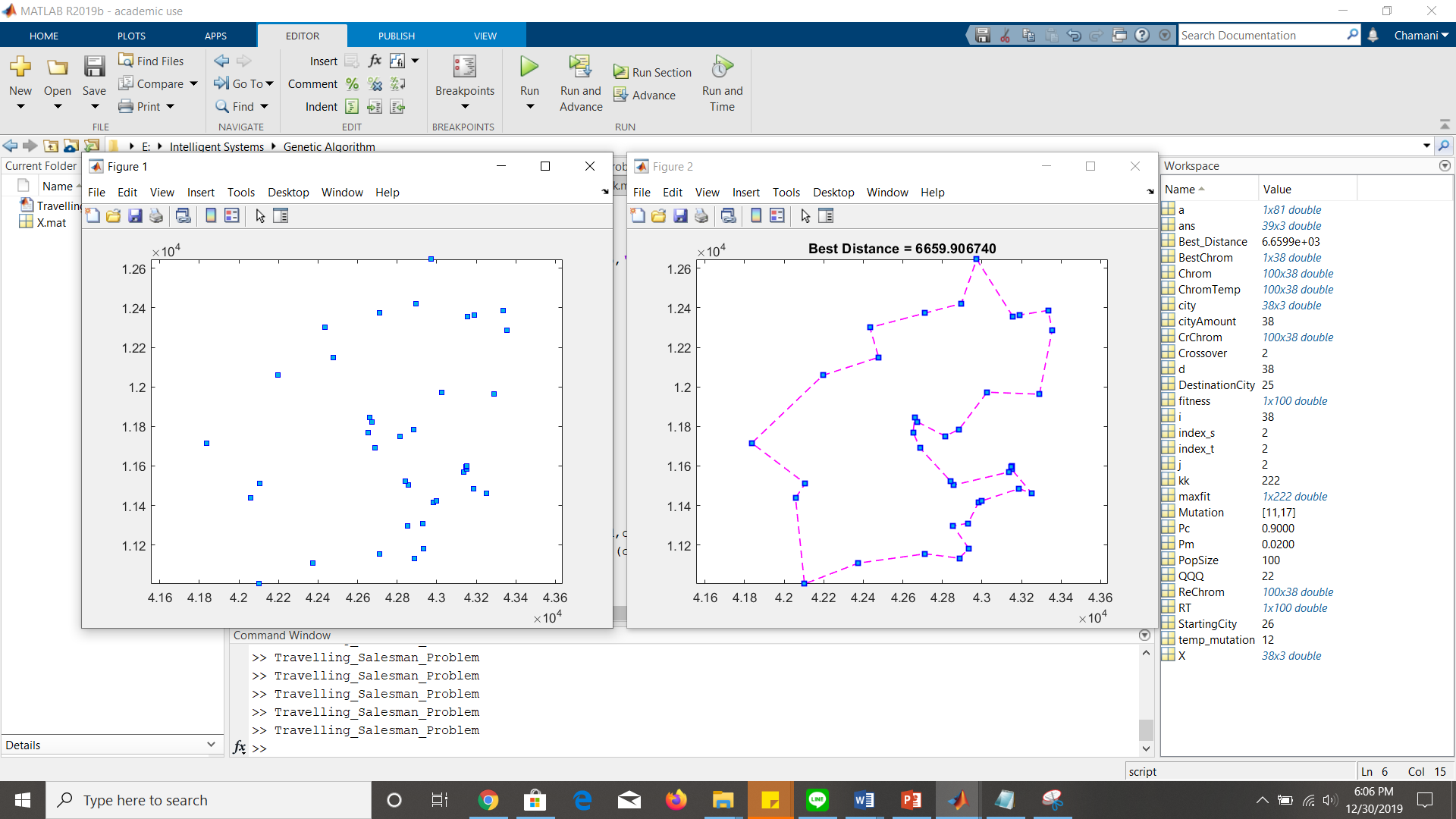
**2nd experiment:**



**3rd experiment:**



**4th experiment: (Best Route)**



These are the variable values at the best route

