A

CASE STUDY AND REPORT

ON

Travelling Salesman Problem Using Genetic Algorithm

MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY JAIPUR

SESSION 2022-23

SUBMITTED TO:

DR. RAJESH KUMAR
ELECTRICAL ENGINEERING
MNIT JAIPUR

SUBMITTED BY:

BIPIN KUMAR
B. TECH 4TH YEAR
2019UEE1340

Travelling Salesman Problem Using Genetic Algorithm

Using a Genetic Algorithm to find a solution to the traveling salesman problem (TSP).

Problem Statement:

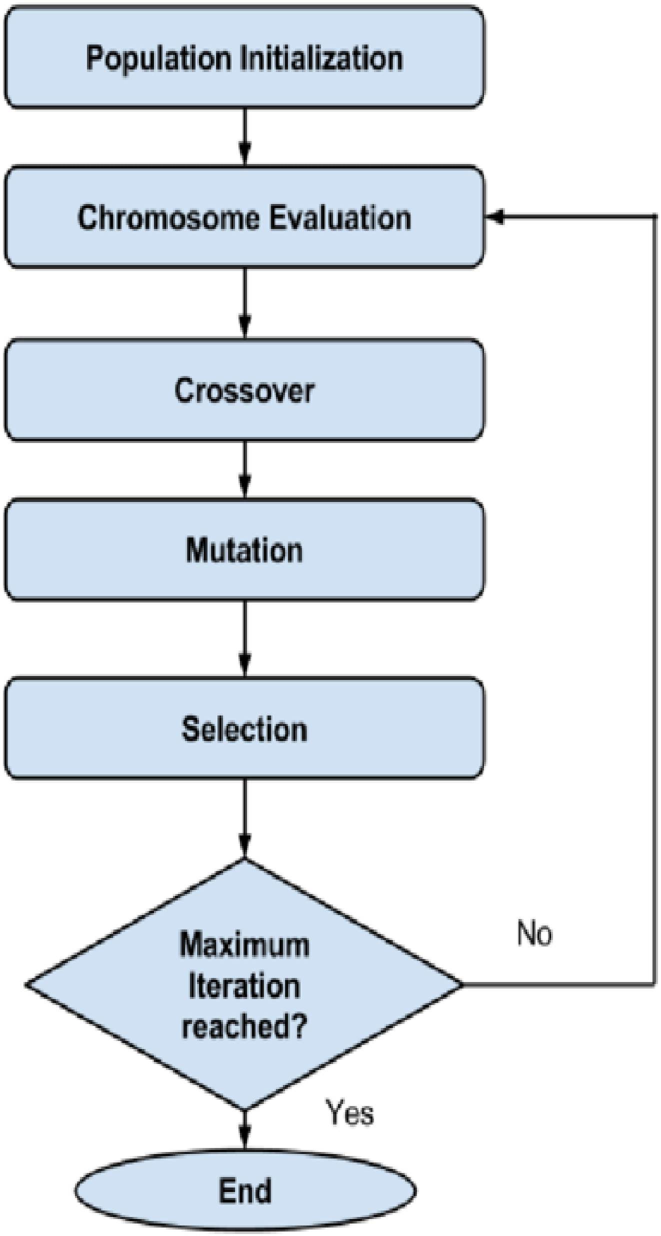
"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city".

Genetic Algorithm:

Genetic algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population . The new population will be better than the old one.Solutions which are selected to form new solutions (offspring) are selected according to their fitness, the more suitable they are the more chances they have to reproduce. This is repeated until some condition for example number of populations or improvement of the best solution is satisfied.

Genetic Algorithm is a paradigm that has proved to be a unique approach for solving various mathematical problems which other gradient type of mathematical optimizers have failed to reach Ant colony optimization has been applied successfully to a large number of difficult combinatorial optimization problems.

FLOWCHART



❖ ALGORITHM PROCESS

Step 1. Create an initial population of P chromosomes.

Step 2. Evaluate the fitness of each chromosome.

Step 3. Choose P/2 parents from the current population via proportional selection.

Step 4. Randomly select two parents to create offspring using crossover operator.

Step 5. Apply mutation operators for minor changes in the results.

Step 6. Repeat Steps 4 and 5 until all parents are selected and mated.

Step 7. Replace old population of chromosomes with new one.

Step 8. Evaluate the fitness of each chromosome in the new population.

Step 9. Terminate if the number of generations meets some upper bound; otherwise go to Step 3.

In [1]:
```
#Importing Libraries

#For Manipulations
import numpy as np
import pandas as pd

#For Data Visualizations
import  matplotlib.pyplot as plt

import random
import operator
import math
```

In [2]:
```
def distance_between_cities(cities):
    data = dict()
    for index, value in enumerate(cities):
        x1 = cities[index][0]
        y1 = cities[index][1]
        if index + 1 <= len(cities)-1:
            x2 = cities[index+1][0]
            y2 = cities[index+1][1]
            xdiff = x2 - x1
            ydiff = y2 - y1
            dst = (xdiff*xdiff + ydiff*ydiff)** 0.5
            data['Distance from city '+ str(index+1) +' to city ' + str(index+2)] = ds
        elif index + 1 > len(cities)-1:
            x2 = cities[0][0]
            y2 = cities[0][1]
            xdiff = x2 - x1
            ydiff = y2 - y1
            dst = (xdiff*xdiff + ydiff*ydiff)** 0.5
            data['Distance from city '+ str(index+1) + ' to city ' + str(index +2 -ler

    return data
```

In [3]:
```
cityList  = [[77.580643,12.972442],[72.88261,19.07283],[77.216721,28.644800],[73.85625
             ,[85.158875,25.612677],[80.9231262,26.8392792],[74.797371,34.083656]]
val = distance_between_cities(cityList).values()
```

In [4]:
```
print(val)
```

```
dict_values([7.699756348069267, 10.507479614123506, 10.6710175094333, 13.345476366874
312, 4.409775601291549, 9.487142456541129, 21.2938948898453])
```

In [5]:
```
def total_distance(cities):
    total = sum(distance_between_cities(cities).values())
    return total
total_distance(cityList)
```

Out[5]:
```
77.41454278617837
```

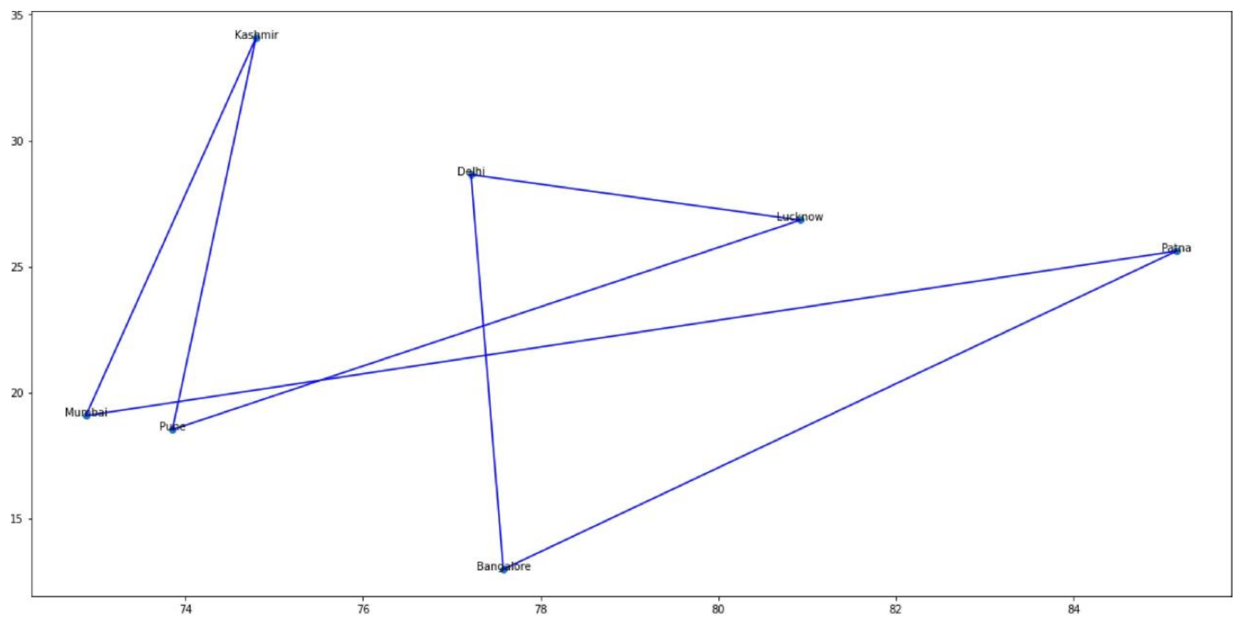
In [6]:
```
def generatePath(cities):
    path = random.sample(cities, len(cities))
    return path
list= generatePath(cityList)
print(list)
```

```
[[72.88261, 19.07283], [74.797371, 34.083656], [73.856255, 18.516726], [80.9231262, 2
6.8392792], [77.216721, 28.6448], [77.580643, 12.972442], [85.158875, 25.612677]]
```

In [7]:
```
import numpy as np
city_names = ['Bangalore', 'Mumbai', 'Delhi', 'Pune','Patna','Lucknow','Kashmir']
def plot_pop(cities):
    plt.figure(figsize=(20,10))
    x = [i[0] for i in cities]
    y = [i[1] for i in cities]
    x1=[x[0],x[-1]]
    y1=[y[0],y[-1]]
    plt.plot(x, y, 'b', x1, y1, 'b')
    plt.scatter (x, y)
    j =[77.580643, 72.88261, 77.216721, 73.856255,85.158875,80.9231262,74.797371]
    k = [12.972442, 19.07283, 28.644800, 18.516726,25.612677,26.8392792,34.083656]


    for i, txt in enumerate(city_names):
        plt.annotate(txt, (j[i], k[i]),horizontalalignment='center',
            #verticalalignment='bottom',
                    )
    plt.show()
    return
```

In [8]:
```
plot_pop(list)
```

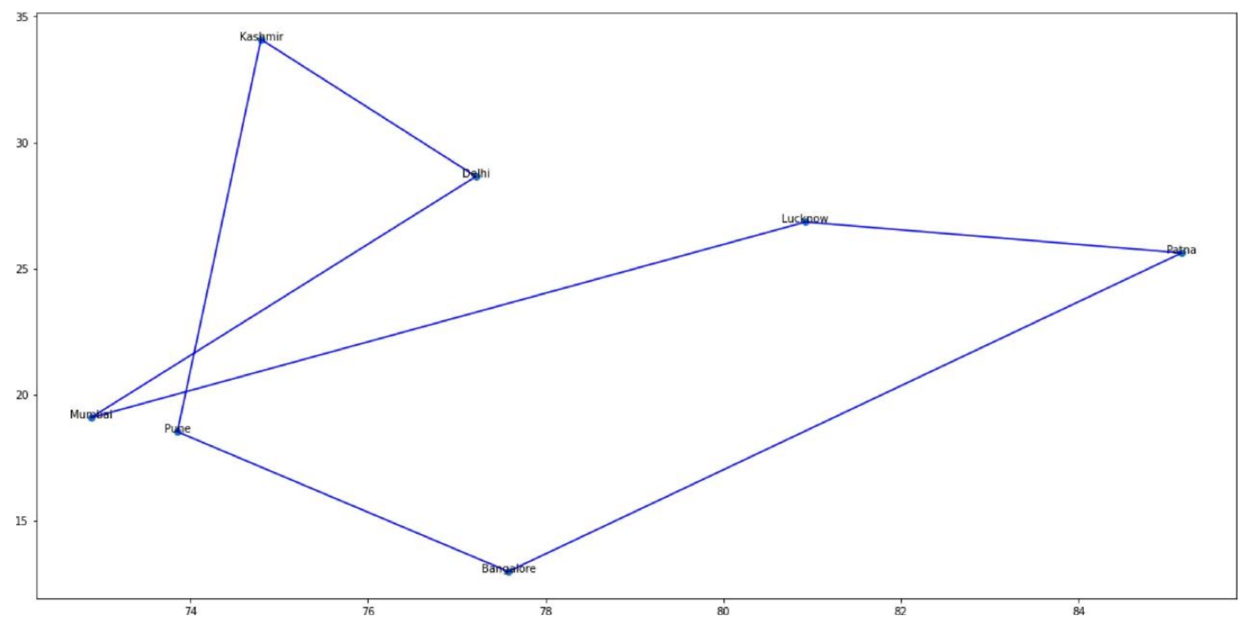


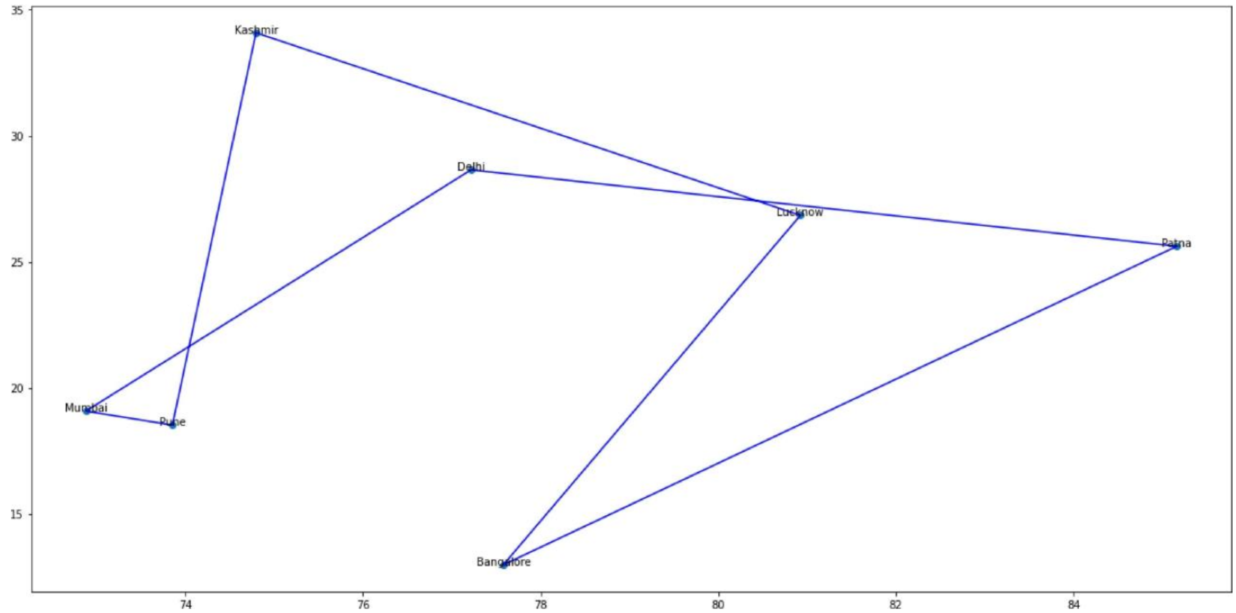
In [9]:
```
def initialPopulation(cities, populationSize):
    population = [generatePath(cities) for i in range(0, populationSize)]
    return population
population = initialPopulation(cityList,10)
```

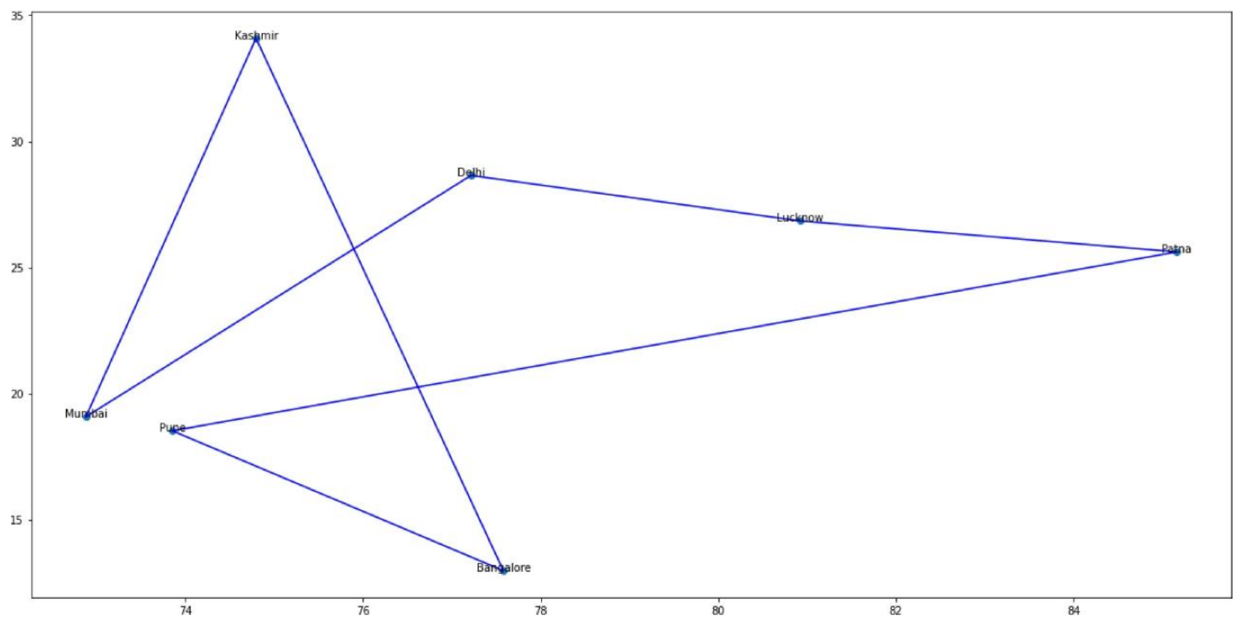
In [10]:
```
for idx, pop_plot in enumerate (population):
    print('Initial Population '+ str(idx),pop_plot)
```

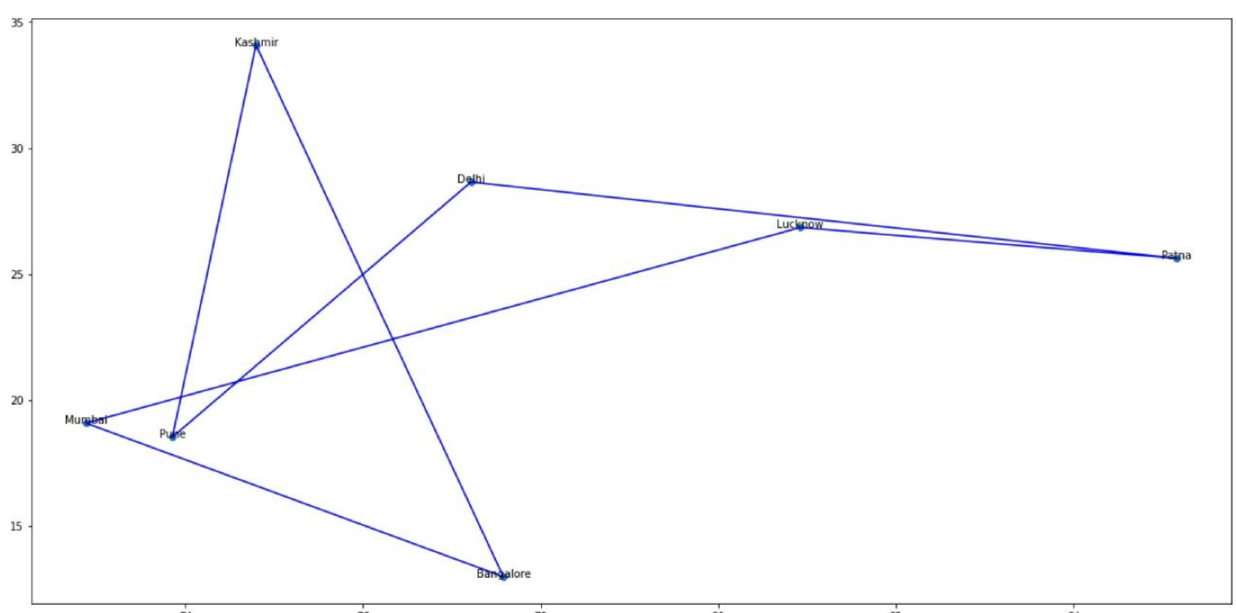
```
Initial Population 0 [[80.9231262, 26.8392792], [85.158875, 25.612677], [77.580643, 1
2.972442], [73.856255, 18.516726], [74.797371, 34.083656], [77.216721, 28.6448], [72.
88261, 19.07283]]
Initial Population 1 [[85.158875, 25.612677], [77.216721, 28.6448], [72.88261, 19.072
83], [73.856255, 18.516726], [74.797371, 34.083656], [80.9231262, 26.8392792], [77.58
0643, 12.972442]]
Initial Population 2 [[72.88261, 19.07283], [74.797371, 34.083656], [77.580643, 12.97
2442], [73.856255, 18.516726], [85.158875, 25.612677], [80.9231262, 26.8392792], [77.
216721, 28.6448]]
Initial Population 3 [[74.797371, 34.083656], [77.580643, 12.972442], [72.88261, 19.0
7283], [80.9231262, 26.8392792], [85.158875, 25.612677], [77.216721, 28.6448], [73.85
6255, 18.516726]]
Initial Population 4 [[77.580643, 12.972442], [73.856255, 18.516726], [85.158875, 25.
612677], [77.216721, 28.6448], [74.797371, 34.083656], [72.88261, 19.07283], [80.9231
262, 26.8392792]]
Initial Population 5 [[74.797371, 34.083656], [77.216721, 28.6448], [80.9231262, 26.8
392792], [73.856255, 18.516726], [85.158875, 25.612677], [77.580643, 12.972442], [72.
88261, 19.07283]]
Initial Population 6 [[73.856255, 18.516726], [74.797371, 34.083656], [77.580643, 12.
972442], [85.158875, 25.612677], [77.216721, 28.6448], [80.9231262, 26.8392792], [72.
88261, 19.07283]]
Initial Population 7 [[77.580643, 12.972442], [73.856255, 18.516726], [85.158875, 25.
612677], [80.9231262, 26.8392792], [77.216721, 28.6448], [74.797371, 34.083656], [72.
88261, 19.07283]]
Initial Population 8 [[85.158875, 25.612677], [74.797371, 34.083656], [77.216721, 28.
6448], [80.9231262, 26.8392792], [72.88261, 19.07283], [77.580643, 12.972442], [73.85
6255, 18.516726]]
Initial Population 9 [[72.88261, 19.07283], [74.797371, 34.083656], [73.856255, 18.51
6726], [80.9231262, 26.8392792], [77.580643, 12.972442], [77.216721, 28.6448], [85.15
8875, 25.612677]]
```

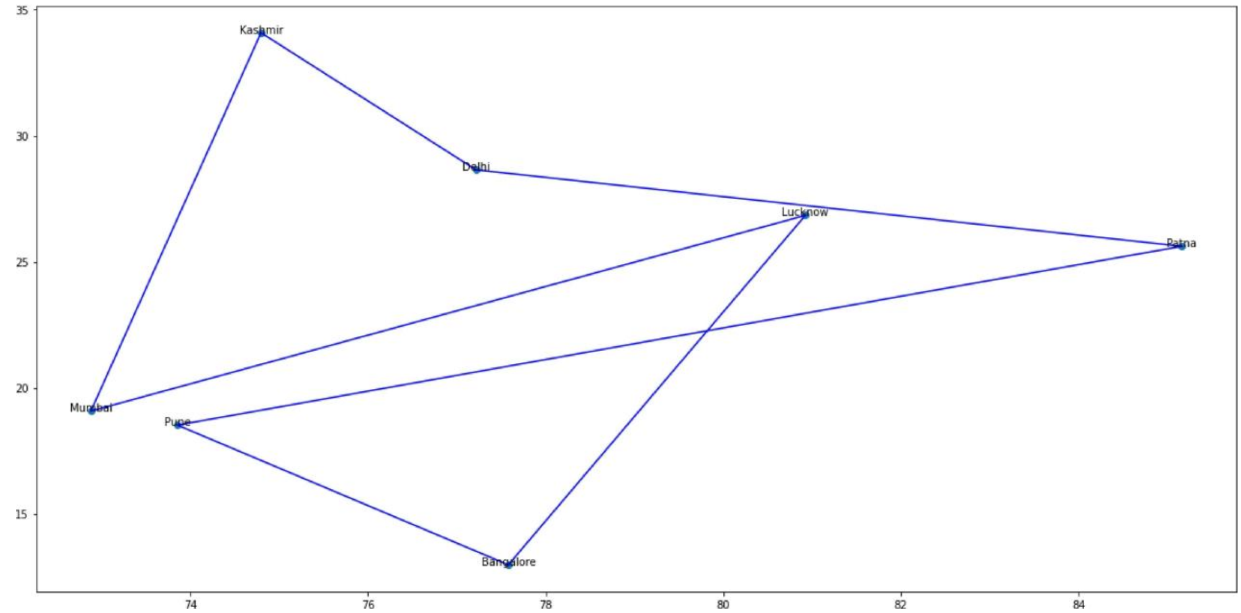
In [11]:
```python
for pop_plot in population:
    plot_pop(pop_plot)
```



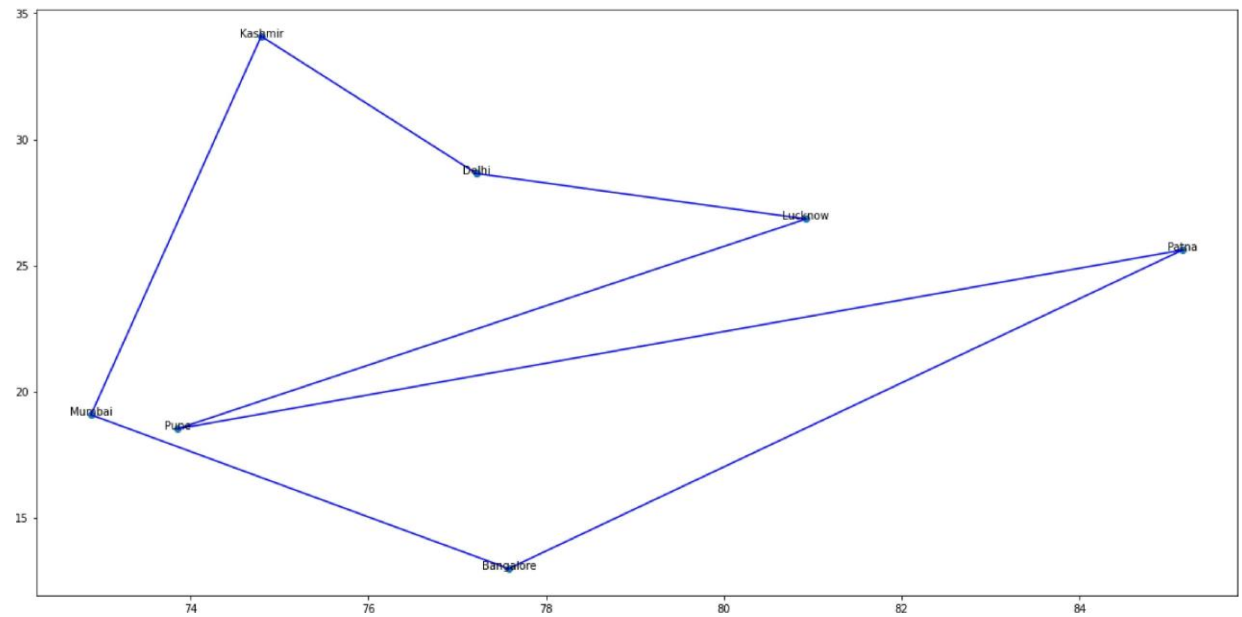

Kashmir
Delhi
Lucknow
Patna
Mumbai
Pune
Bangalore
35
30
25
20
15
74
76
78
80
82
84


Kashmir
Delhi
Lucknow
Patna
Mumbai
Pune
Bangalore
35
30
25
20
15
74
76
78
80
82
84

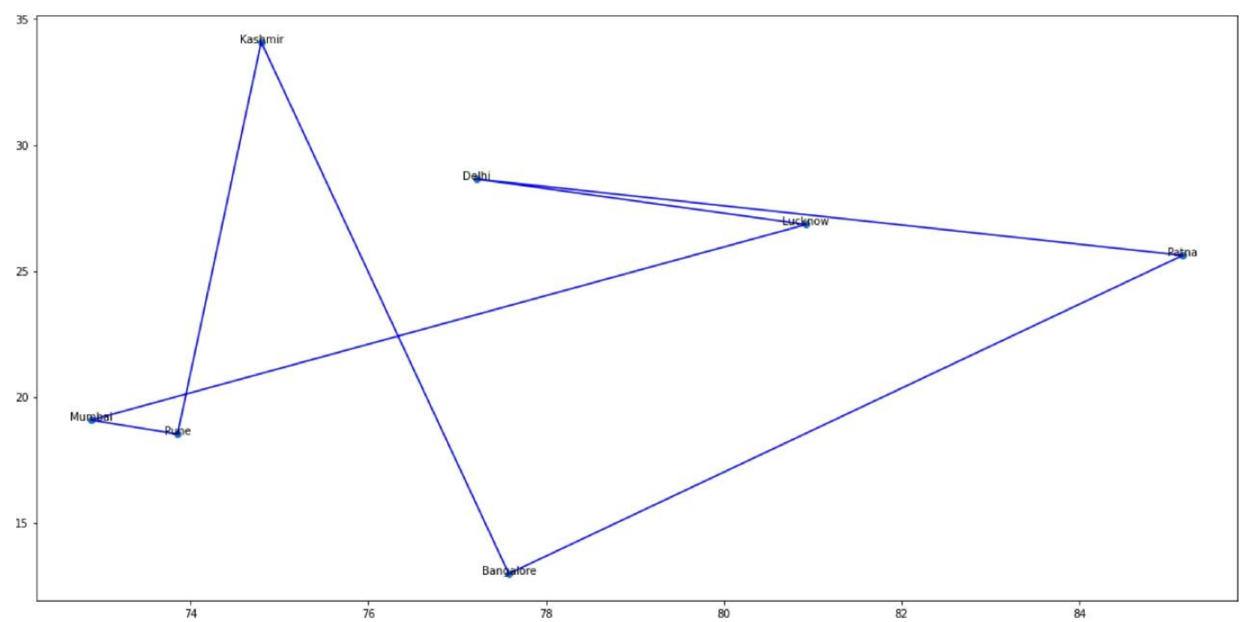

Kashmir
Delhi
Lucknow
Patna
Mumbai
Pune
Bangalore
35
30
25
20
15
74
76
78
80
82
84



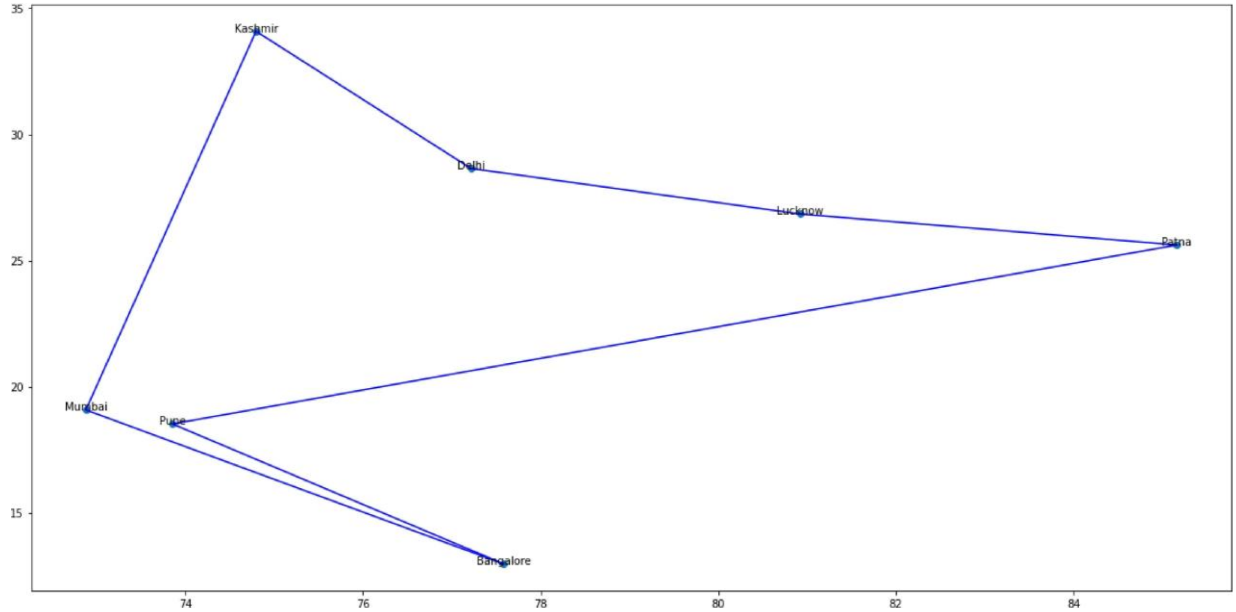
Kashmir
Delhi
Lucknow
Patna
Mumbai
Pune
Bangalore

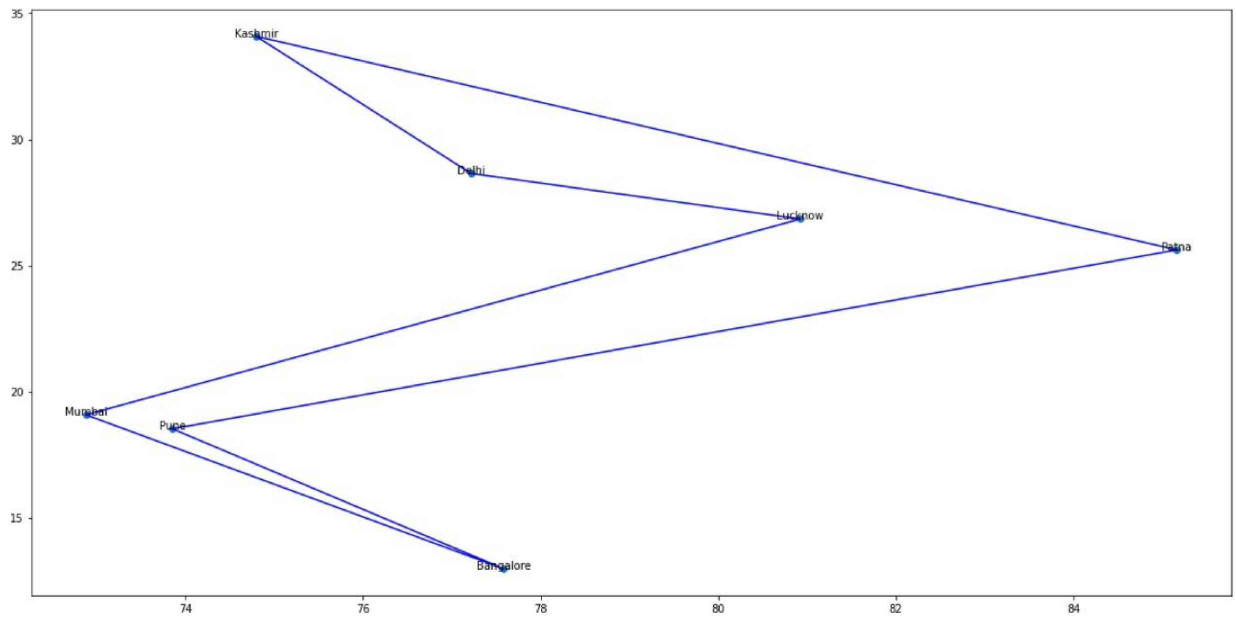


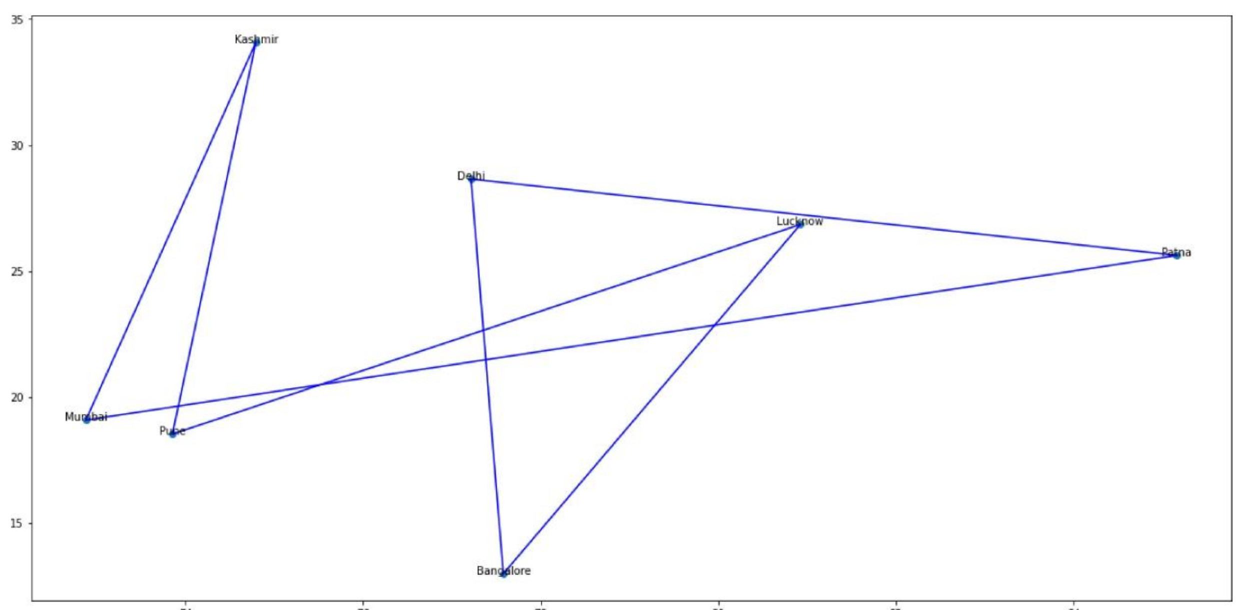
Kashmir
Delhi
Lucknow
Patna
Mumbai
Pune
Bangalore



Kashmir
Delhi
Lucknow
Patna
Mumbai
Pune
Bangalore







In [12]:
```
def path_fitness(cities):
```

```
    total_dis = total_distance(cities)
    fitness= 0.0
    if fitness == 0:
        fitness = 1 / float(total_dis)
    return fitness
path_fitness(cityList)
```

Out[12]: 0.012917469560752097

In [13]:
```
def rankPathes(population):
    fitnessResults = {}
    for i in range(len(population)):
        fitnessResults[i] = path_fitness(population[i])

    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True
rankPathes(population)
```

Out[13]:
```
[(7, 0.017439220870653156),
 (8, 0.016035360082012715),
 (0, 0.014479922145976291),
 (5, 0.013906433469754268),
 (1, 0.013474477828625042),
 (4, 0.013323767602923808),
 (2, 0.013246621195215812),
 (6, 0.013063128790737946),
 (3, 0.012602401098961668),
 (9, 0.010638597436880743)]
```

In [14]:
```
def perform_selection(pop, eliteSize):
    #output = rankPathes(population)
    df = pd.DataFrame(np.array(pop), columns=["Index","Fitness"])
#A cumulative sum is a sequence of partial sums of a given sequence
    df['cumulative_sum'] = df.Fitness.cumsum()
#Cumulative percentage is another way of expressing frequency distribution.
#It calculates the percentage of the cumulative frequency within each interval, much a
    df['cum_percentage'] = 100*df.cumulative_sum/df.Fitness.sum()
    selected_values = [pop[i][0] for i in range(eliteSize)]

    for i in range(len(pop) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(pop)):
            if pick <= df.iat[i,3]:
                selected_values.append(pop[i][0])
                break

    return selected_values
```

In [15]:
```
out11 = rankPathes(population)
selected_values = perform_selection(out11,5)
print(selected_values)
```

```
[7, 8, 0, 5, 1, 7, 0, 6, 9, 9]
```

In [16]:
```
def do_mating_pool(population, selected_values):
    matingpool = [population[selected_values[i]] for i in range(len(selected_values))]
    return matingpool
mp = do_mating_pool(population, selected_values)
```

In [17]:
```
def do_breed(first_parent, second_parent):
```

```
    generation_1= int(random.random() * len(first_parent))
    generation_2 = int(random.random() * len(second_parent))

    first_generation = min(generation_1, generation_2)
    last_generation = max(generation_1, generation_2)

    tot_parent1 = [first_parent[i] for i in range(first_generation, last_generation)]
    tot_parent2 = [i for i in second_parent if i not in tot_parent1]

    tot = tot_parent1 + tot_parent2
    return tot
```

In [18]:

```
def do_breed_population(my_mating_pool, eliteSize):
    ln = len(my_mating_pool) - eliteSize
    pl = random.sample(my_mating_pool, len(my_mating_pool))
    tot1 = [my_mating_pool[i] for i in range(eliteSize)]
    tot2 = [do_breed(pl[i], pl[len(my_mating_pool)-i-1]) for i in range(ln)]
    tot = tot1+tot2
    return tot
do_breed_population(mp,2)
```

Out[18]:
```
[[[77.580643, 12.972442],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [80.9231262, 26.8392792],
  [77.216721, 28.6448],
  [74.797371, 34.083656],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283],
  [77.580643, 12.972442],
  [73.856255, 18.516726]],
 [[77.216721, 28.6448],
  [72.88261, 19.07283],
  [74.797371, 34.083656],
  [73.856255, 18.516726],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442],
  [85.158875, 25.612677]],
 [[77.216721, 28.6448],
  [74.797371, 34.083656],
  [77.580643, 12.972442],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283],
  [77.580643, 12.972442],
  [73.856255, 18.516726]],
 [[73.856255, 18.516726],
  [74.797371, 34.083656],
  [77.580643, 12.972442],
  [85.158875, 25.612677],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283]],
 [[74.797371, 34.083656],
  [77.216721, 28.6448],
  [85.158875, 25.612677],
  [72.88261, 19.07283],
  [73.856255, 18.516726],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442]],
 [[74.797371, 34.083656],
  [80.9231262, 26.8392792],
  [77.216721, 28.6448],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [77.580643, 12.972442],
  [72.88261, 19.07283]],
 [[74.797371, 34.083656],
  [77.580643, 12.972442],
  [85.158875, 25.612677],
  [77.216721, 28.6448],
```

```
  [80.9231262, 26.8392792],
  [72.88261, 19.07283],
  [73.856255, 18.516726]],
 [[85.158875, 25.612677],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283],
  [77.580643, 12.972442],
  [73.856255, 18.516726]]]
```

In [19]:

```
def do_mutatation(indiv, mutat_rate):
    for exchanged in range(len(indiv)):
        if(random.random() < mutat_rate):
            exchanged_with = int(random.random() * len(indiv))

            city1 = indiv[exchanged]
            city2 = indiv[exchanged_with]

            indiv[exchanged] = city2
            indiv[exchanged_with] = city1
    return indiv
```

In [20]:

```
def do_mutatation_pop(population, mutat_rate):
    mutated_population = [do_mutatation(population[i], mutat_rate) for i in range(len(
    return mutated_population
do_mutatation_pop(population, 0.01)
```

```
Out[20]: [[[80.9231262, 26.8392792],
  [85.158875, 25.612677],
  [77.580643, 12.972442],
  [73.856255, 18.516726],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [77.216721, 28.6448],
  [72.88261, 19.07283],
  [73.856255, 18.516726],
  [74.797371, 34.083656],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442]],
 [[72.88261, 19.07283],
  [74.797371, 34.083656],
  [77.580643, 12.972442],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [80.9231262, 26.8392792],
  [77.216721, 28.6448]],
 [[74.797371, 34.083656],
  [77.580643, 12.972442],
  [72.88261, 19.07283],
  [80.9231262, 26.8392792],
  [85.158875, 25.612677],
  [77.216721, 28.6448],
  [73.856255, 18.516726]],
 [[77.580643, 12.972442],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [77.216721, 28.6448],
  [74.797371, 34.083656],
  [72.88261, 19.07283],
  [80.9231262, 26.8392792]],
 [[74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442],
  [85.158875, 25.612677],
  [73.856255, 18.516726],
  [72.88261, 19.07283]],
 [[73.856255, 18.516726],
  [74.797371, 34.083656],
  [77.580643, 12.972442],
  [85.158875, 25.612677],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283]],
 [[77.580643, 12.972442],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [80.9231262, 26.8392792],
  [77.216721, 28.6448],
  [74.797371, 34.083656],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
```

```
  [72.88261, 19.07283],
  [77.580643, 12.972442],
  [73.856255, 18.516726]],
 [[72.88261, 19.07283],
  [74.797371, 34.083656],
  [73.856255, 18.516726],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442],
  [77.216721, 28.6448],
  [85.158875, 25.612677]]]
```

In [21]:
```
def get_following_gen(existing_gen, eliteSize, mutat_rate):
    pop = rankPathes(existing_gen)

    selected_values = perform_selection(pop, eliteSize)

    my_mating_pool = do_mating_pool(existing_gen, selected_values)
    tot = do_breed_population(my_mating_pool, eliteSize)
    following_gen = do_mutatation(tot, mutat_rate)
    #print(following_gen)
    return following_gen
get_following_gen(population, 5, 0.01)
```

Out[21]:
```
[[[77.580643, 12.972442],
  [73.856255, 18.516726],
  [85.158875, 25.612677],
  [80.9231262, 26.8392792],
  [77.216721, 28.6448],
  [74.797371, 34.083656],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283],
  [77.580643, 12.972442],
  [73.856255, 18.516726]],
 [[74.797371, 34.083656],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442],
  [85.158875, 25.612677],
  [73.856255, 18.516726],
  [72.88261, 19.07283]],
 [[80.9231262, 26.8392792],
  [85.158875, 25.612677],
  [77.580643, 12.972442],
  [73.856255, 18.516726],
  [74.797371, 34.083656],
  [77.216721, 28.6448],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [77.216721, 28.6448],
  [72.88261, 19.07283],
  [73.856255, 18.516726],
  [74.797371, 34.083656],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442]],
 [[73.856255, 18.516726],
  [74.797371, 34.083656],
  [77.580643, 12.972442],
  [85.158875, 25.612677],
  [77.216721, 28.6448],
  [80.9231262, 26.8392792],
  [72.88261, 19.07283]],
 [[85.158875, 25.612677],
  [77.216721, 28.6448],
  [72.88261, 19.07283],
  [73.856255, 18.516726],
  [74.797371, 34.083656],
  [80.9231262, 26.8392792],
  [77.580643, 12.972442]],
 [[77.580643, 12.972442],
  [85.158875, 25.612677],
  [73.856255, 18.516726],
  [77.216721, 28.6448],
  [74.797371, 34.083656],
  [72.88261, 19.07283],
  [80.9231262, 26.8392792]],
 [[85.158875, 25.612677],
  [80.9231262, 26.8392792],
  [77.216721, 28.6448],
  [74.797371, 34.083656],
```

```
 [72.88261, 19.07283],
 [77.580643, 12.972442],
 [73.856255, 18.516726]],
[[80.9231262, 26.8392792],
 [77.216721, 28.6448],
 [74.797371, 34.083656],
 [85.158875, 25.612677],
 [77.580643, 12.972442],
 [73.856255, 18.516726],
 [72.88261, 19.07283]]]
```

In [22]:

```
#cityList  =  [[77.580643,12.972442],[72.88261,19.07283],[77.216721,28.644800],[73.856
             #,[85.158875,25.612677],[80.9231262,26.8392792],[74.797371,34.083656]]
#city_names =['Bangalore', 'Mumbai', 'Delhi', 'Pune','Patna','Lucknow','Kashmir']
def get_names(result_lst, cities, name_lst):
    names = []
    for index,value in enumerate(result_lst):
        for i,v in enumerate(cities):
            if value == v:
                names.append(name_lst[i])
    return names
```

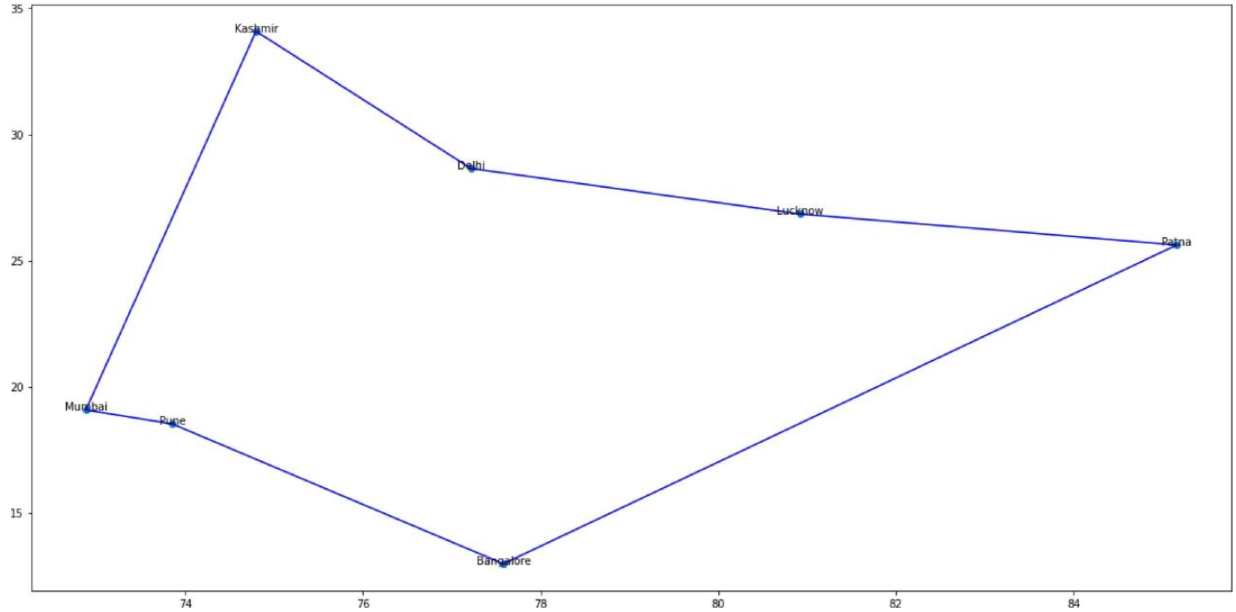
In [23]:

```
def GA(city_names,cities, population_size, eliteSize, mutat_rate, generations):
    population = initialPopulation(cities,population_size)
    #print(population_)
    print("Incipient distance: " + str(1 / rankPathes(population)[0][1]))
    for i in range(generations):
        population = get_following_gen(population, eliteSize, mutat_rate)
        #print(population)

    print("Eventual distance: " + str(1 / rankPathes(population)[0][1]))
    optimal_route_id = rankPathes(population)[0][0]
    optimal_route = population[optimal_route_id]
    ordered_cities = get_names(optimal_route,cities,city_names)
    print([(indx,val) for indx,val in enumerate(ordered_cities)])
    plot_pop(optimal_route)
    return optimal_route

result_lst = GA(city_names,cityList, population_size=100,
                eliteSize=5, mutat_rate=0.01,
                generations=500)
```

```
Incipient distance: 52.895308223146436
Eventual distance: 52.15592519829276
[(0, 'Lucknow'), (1, 'Patna'), (2, 'Bangalore'), (3, 'Pune'), (4, 'Mumbai'), (5, 'Kas
hmir'), (6, 'Delhi')]
```



In [24]:
```
print(result_lst)
```

```
[[80.9231262, 26.8392792], [85.158875, 25.612677], [77.580643, 12.972442], [73.85625
5, 18.516726], [72.88261, 19.07283], [74.797371, 34.083656], [77.216721, 28.6448]]
```

Conclusion :

Various crossover operators have been presented for TSP with different applications by using Genetic Algorithm. The traveling salesman problem (TSP) has commanded much attention from mathematicians and computer scientists specifically because it is so easy to describe and so difficult to solve. The problem can simply be stated as: if a traveling salesman wishes to visit exactly once each of a list of m cities (where the cost of traveling from city i to city j is c_{ij}) and then return to the home city, what is the least costly route the traveling salesman can take?