

# Simulated Annealing

Travelling salesman problem using simulated annealing

Ankur Singh  
MTECH CSE (2020-22)  
IIT Vadodara  
Gandhinagar, India  
202061001@iitvadodara.ac.in

Anand Mundhe  
MTECH CSE (2020-22)  
IIT Vadodara  
Gandhinagar, India  
202061006@iitvadodara.ac.in

## I. PROBLEM STATEMENT

A. Travelling Salesman Problem (TSP) is a hard problem, and is simple to state. Given a graph in which the nodes are locations of cities, and edges are labelled with the cost of travelling between cities, find a cycle containing each city exactly once, such that the total cost of the tour is as low as possible. For the state of Rajasthan, find out at least twenty important tourist locations. Suppose your relatives are about to visit you next week. Use Simulated Annealing to plan a cost effective tour of Rajasthan. It is reasonable to assume that the cost of travelling between two locations is proportional to the distance between them.

B. An interesting problem domain with TSP instances: VLSI: <http://www.math.uwaterloo.ca/tsp/vlsi/index.html#XQF131> (Attempt at least five problems from the above list and compare your results.)

## II. SOLUTION OF TSP IN RAJASTHAN

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from tqdm import notebook
4 class coordinate:
5     def __init__(self,x,y):
6         self.x=x
7         self.y=y
8
9     @staticmethod
10    def get_distance(a,b):
11        return np.sqrt(np.abs(a.x-b.x)**2+np.abs(a.y
12        -b.y)**2)
13
14    @staticmethod
15    def get_total_distance(coords):
16        dist=0
17        for first,second in zip(coords[:-1],coords
18        [1:]):
19            dist+=coordinate.get_distance(first,
20            second)
21        dist+=coordinate.get_distance(coords[0],
22        coords[-1])
23        return dist
24
25 def readTSP(f):
26     f1=open(f)
27     s=""
28     flag=False
29     coords=[]
30     s=f1.readline()
31     while s!="EOF\n":
```

```
28     if s=="NODE_COORD_SECTION\n":flag=True
29     elif flag:
30         s=s.split()
31         coords.append(coordinate(int(s[1]),int(s
32         [2])))
33         s=f1.readline()
34     return coords
35
36 coords=readTSP("rajds.tsp")
37 f=open("sortedLocation")
38 s=""
39 s=f.readline()
40 while(s!=""):
41     print(s,end=" ")
42     s=f.readline()
43
44 import matplotlib.pyplot as plt
45 import matplotlib.image as mpimg
46 plt.figure(figsize=(10,10))
47 plt.title("100px is 50KM")
48 img = mpimg.imread('map RAJ.png')
49 imgplot = plt.imshow(img)
50 plt.show()
51
52 def driver(f,flag=False):
53     #read and plot initial coordinates
54     print("Working with :",f)
55     coords=readTSP(f)
56     plt.figure(figsize=(10,10))
57     plt.title("Before TSP")
58     for first,second in zip(coords[:-1],coords[1:]):
59         plt.plot([first.x,second.x],[first.y,second.
60         y], 'b')
61     plt.plot([coords[0].x,coords[-1].x],[coords[0].y
62     ,coords[-1].y], 'b')
63     for c in coords:
64         plt.plot(c.x,c.y, 'ro')
65     plt.grid(True)
66     print("Cost is : ",coordinate.get_total_distance
67     (coords))
68     if flag:plt.gca().invert_yaxis()
69     plt.show()
70     #Apply TSP
71     from sys import stdout
72     from time import sleep
73     cost0=coordinate.get_total_distance(coords)
74     T=30
75     factor=0.999
76     T_init=T
77     epochs=10000
78     for i in notebook.tqdm(range(epochs),total=
79     epochs,unit="epoch"):
80         stdout.write("\rNew cost is %f" % cost0)
81         stdout.flush()
82         #sleep(0.5)
```

```

78     T=T*factor
79     for j in range(100):
80         r1,r2=np.random.randint(0,len(coords),
size=2)
81         temp=coords[r1]
82         coords[r1]=coords[r2]
83         coords[r2]=temp
84         cost1=coordinate.get_total_distance(
coords)
85         if cost1<cost0:
86             cost0=cost1
87         else:
88             x=np.random.uniform()
89             if x<np.exp((cost0-cost1)/T):
90                 cost0=cost1
91             else:
92                 temp=coords[r1]
93                 coords[r1]=coords[r2]
94                 coords[r2]=temp
95     #Plot obtained coordinates
96     plt.figure(figsize=(10,10))
97     plt.title("After TSP")
98     for first,second in zip(coords[:-1],coords[1:]):
99         plt.plot([first.x,second.x],[first.y,second.
y],'b')
100    plt.plot([coords[0].x,coords[-1].x],[coords[0].y
,coords[-1].y],'b')
101    for c in coords:
102        plt.plot(c.x,c.y,'ro')
103    plt.grid(True)
104    if flag:plt.gca().invert_yaxis()
105    plt.show()
106
107 driver("rajds.tsp",True)

```

Listing 1. TSP using simulated annealing in Rajasthan

## Output

```

Ajmer 148 314
beawar 90 385
Bijainagar 150 420
Chomu 344 168
Dausa 455 292
Didwana 136 120
Jaipur 356 221
Kishangarh 187 287
Kuchaman_city 187 173
Makrana 162 197
Malpura 281 345
Merta 38 274
Nasirabad 165 345
Navalgudh 260 29
Neem_ka_Thana 355 51
Niwai 383 332
Sawai_Madhopur 461 404
Sikar 239 77
Sujangarh 115 58
Tonk 358 373

```

Fig. 1. Cities in Rajasthan with coordinates



Fig. 2. Cities in Rajasthan on map

The plot titled "Before TSP" displays 15 red data points on a grid. The x-axis ranges from 0 to 500, and the y-axis ranges from 0 to 450, with values increasing downwards. The points are connected by blue lines, forming a single continuous path that visits every point exactly once, representing a solution to the Traveling Salesman Problem before any optimization.

New cost is 1698.701947

The graph shows a path on a grid. The x-axis ranges from 0 to 500, and the y-axis ranges from 0 to 450. The path starts at (50, 275) and ends at (50, 275), visiting 16 points in total. The path is blue with red dots at each point.

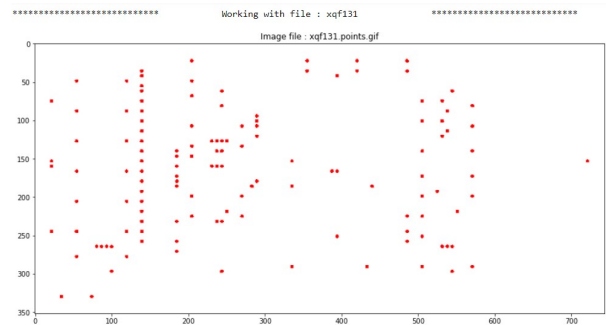
Point	X	Y
1	50	275
2	100	380
3	150	420
4	180	340
5	180	280
6	150	200
7	120	120
8	100	60
9	250	75
10	270	30
11	350	60
12	380	330
13	450	290
14	450	400
15	350	170
16	350	50
17	50	275

### III. SOLUTION OF TSP OF VLSI DATASETS

```
12     image(f+ ".tour.gif")
13
14 file1="xqf131"
15 file2="xqg237"
16 file3="pma343"
17 file4="pka379"
18 file5="bcl1380"
19 files=[file1,file2,file3,file4,file5]
```

1) VLSI dataset 1

```
1 #Apply algorithm for image 1
2 i=files[0]
3 print("\n\n*****\t
\tWorking with file :",i,"\t\t
*****")
4 pimage(i)
5 driver(i+".tsp")
6 qimage(i)
7
```



```
Working with : xqf131.tsp
Cost is : 1383.916876912113
```

The plot titled "Before TSP" displays a time series with a clear upward trend and high volatility. The x-axis ranges from 0 to 110, and the y-axis ranges from 0 to 45. The data points are red dots connected by a blue line, showing frequent sharp increases and decreases. A solid blue linear regression line is overlaid, indicating a positive slope of approximately 0.25 units per x-axis unit.

Fig. 6. Before applying TSP

New cost is 691.8130670

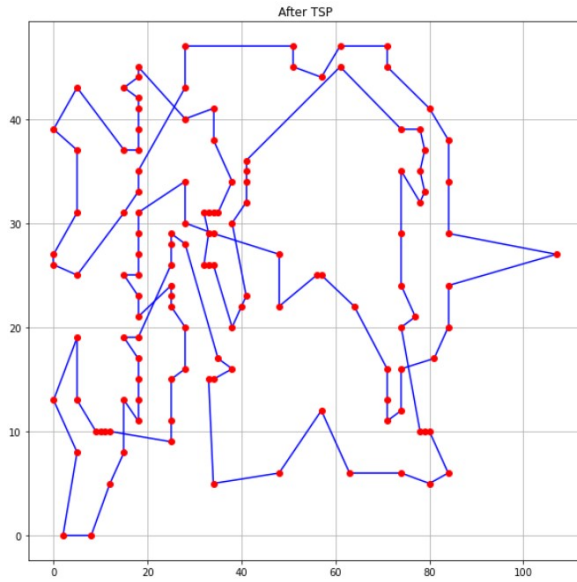


Fig. 7. After applying TSP

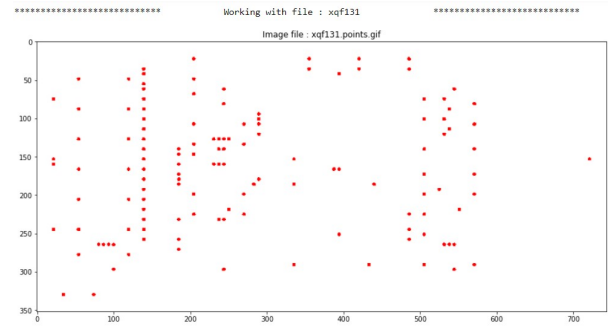


Fig. 9. VLSI dataset 2 [2]

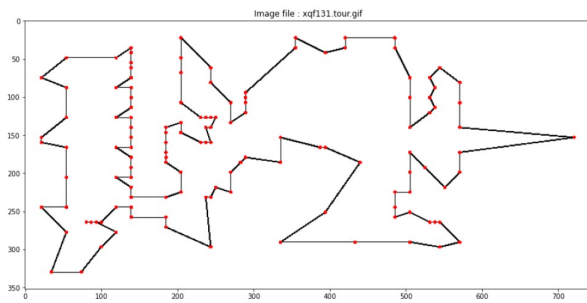


Fig. 8. Actual result [2]

## 2) VLSI dataset 2

```
1 #Apply algorithm for image 2
2 i=files[1]
3 print("\n\n*****\t
  \tWorking with file :",i,"\t\t
  *****")
4 pimage(i)
5 driver(i+".tsp")
6 qimage(i)
7
```

Working with : xqg237.tsp  
Cost is : 2949.5790533940426

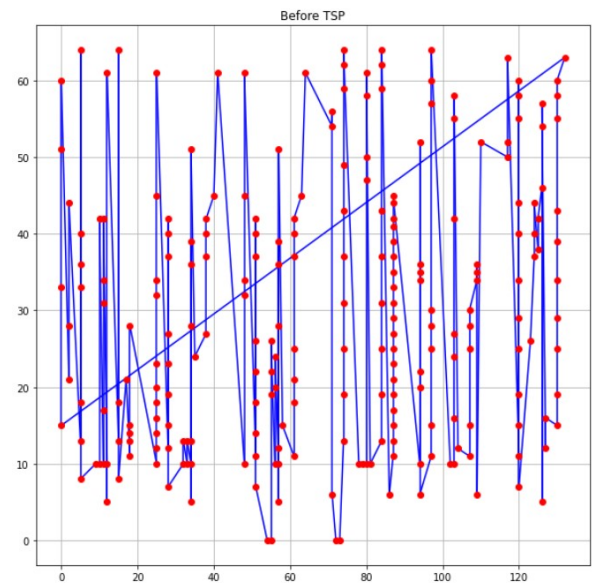


Fig. 10. Before applying TSP

New cost is 1667.093411

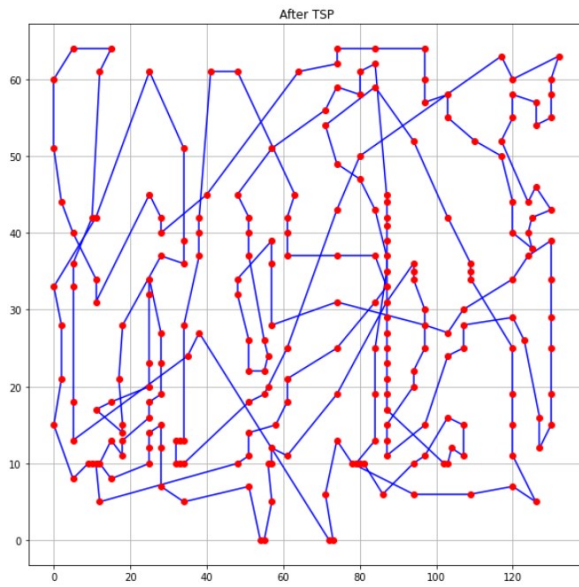


Fig. 11. After applying TSP

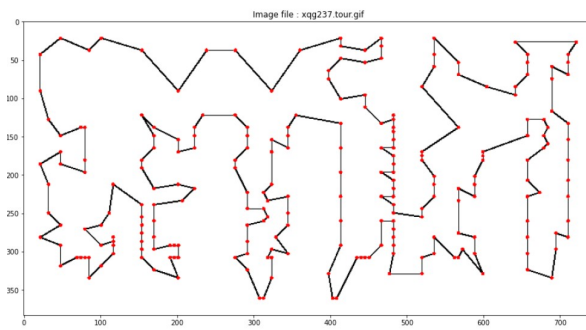


Fig. 12. Actual result [2]



### 3) VLSI dataset 3

```

1 #Apply algorithm for image 3
2 i=files[2]
3 print("\n\n*****\t
\tWorking with file :",i,"\t\t
*****")
4 pimage(i)
5 driver(i+".tsp")
6 qimage(i)
7

```

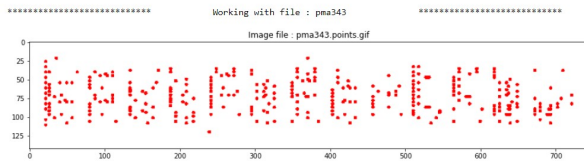


Fig. 13. VLSI dataset 3 [2]

Working with : pma343.tsp  
Cost is : 3117.1798732795

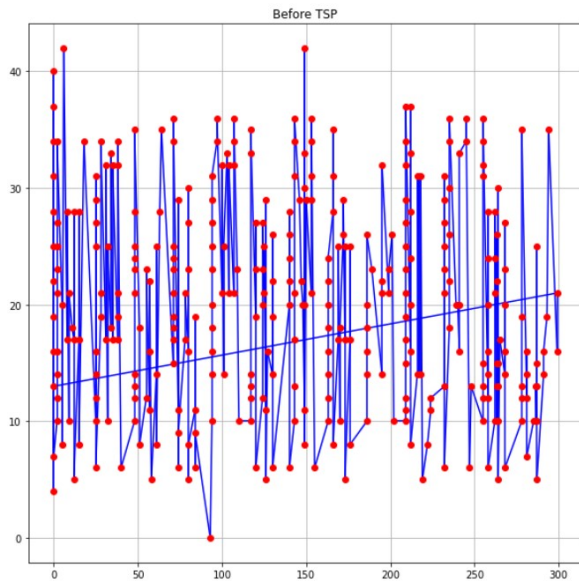


Fig. 14. Before applying TSP

New cost is 2167.790007

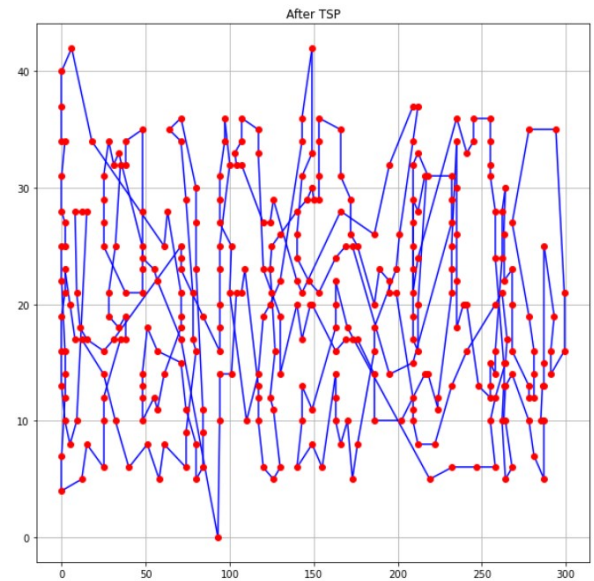


Fig. 15. After applying TSP

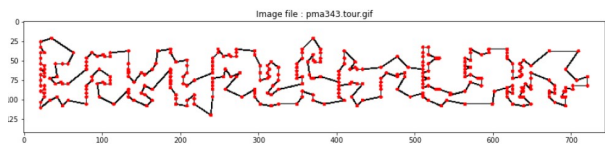


Fig. 16. Actual result [2]

### 4) VLSI dataset 4

```

1 #Apply algorithm for image 4
2 i=files[3]
3 print("\n\n*****\t
\tWorking with file :",i,"\t\t
*****")
4 pimage(i)
5 driver(i+".tsp")
6 qimage(i)
7

```

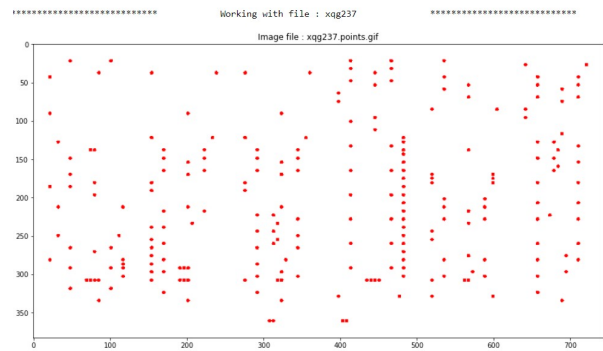


Fig. 17. VLSI dataset 4 [2]

Working with : xqg237.tsp  
 Cost is : 2949.5790533940426

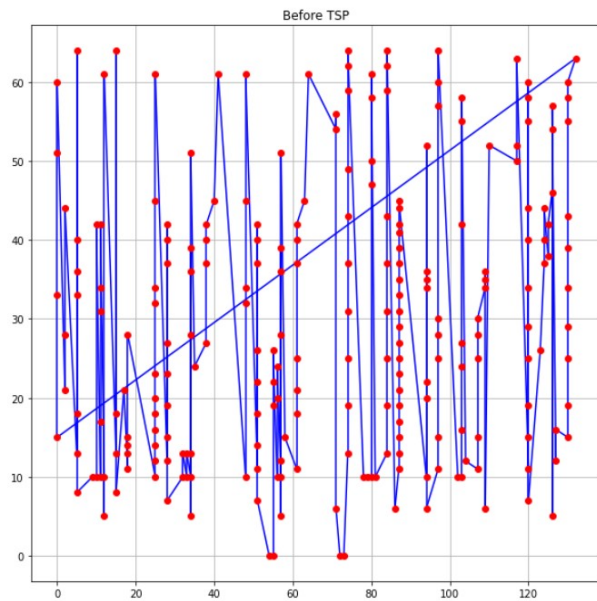


Fig. 18. Before applying TSP

New cost is 1667.093411

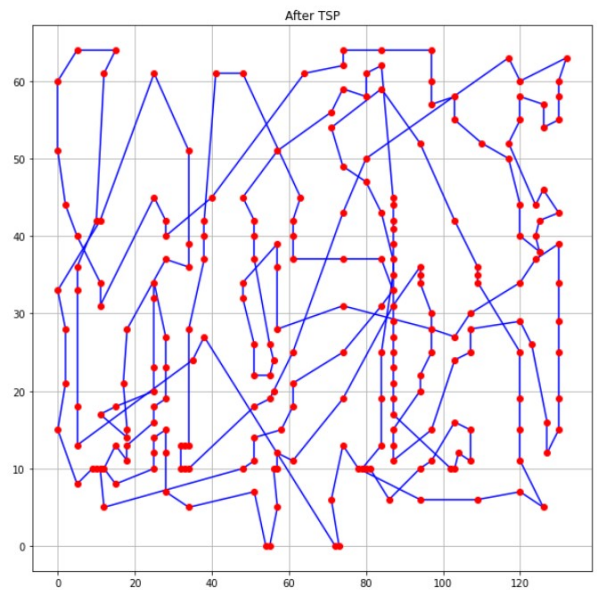


Fig. 19. After applying TSP

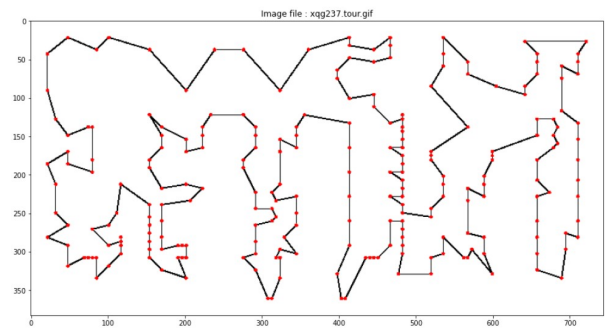


Fig. 20. Actual result [2]

## 5) VLSI dataset 5

```

1 #Apply algorithm for image 5
2 i=files[4]
3 print("\n\n*****\t
  \tWorking with file :",i,"\t\t
  *****")
4 pimage(i)
5 driver(i+".tsp")
6 qimage(i)
7

```

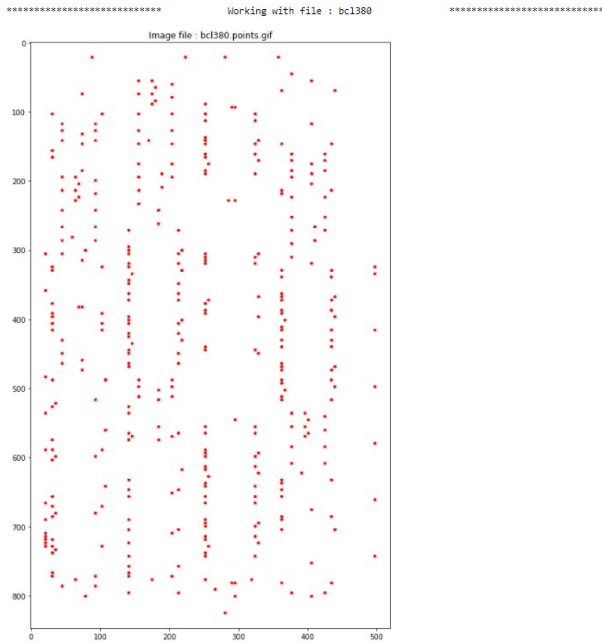


Fig. 21. VLSI dataset 5 [2]

Working with : bcl380.tsp  
Cost is : 10013.54006898525

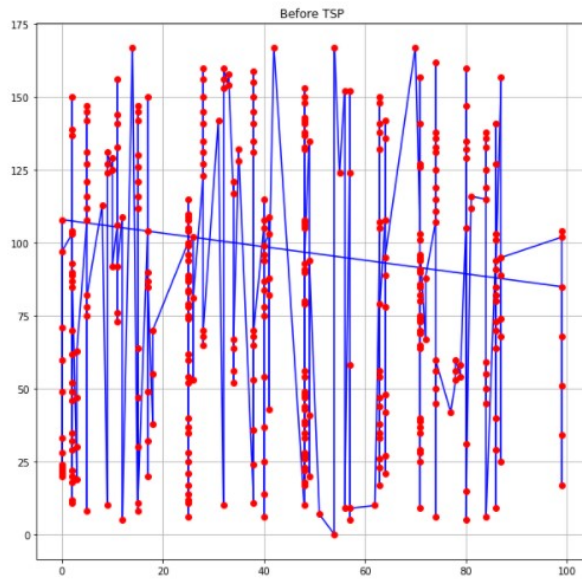


Fig. 22. Before applying TSP

New cost is 3262.3050763

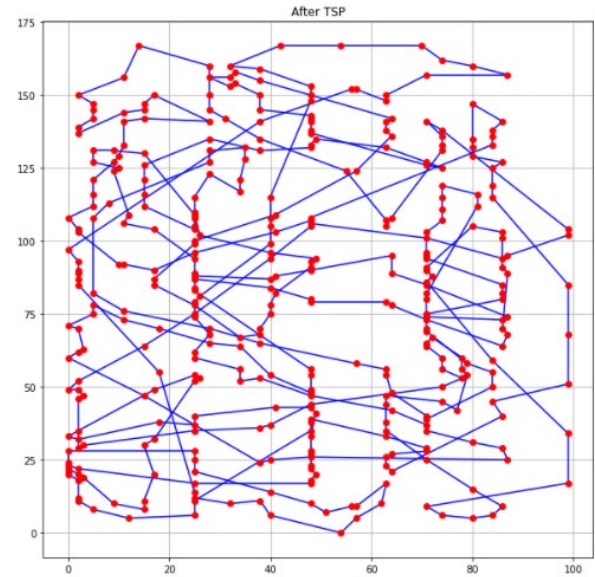


Fig. 23. After applying TSP

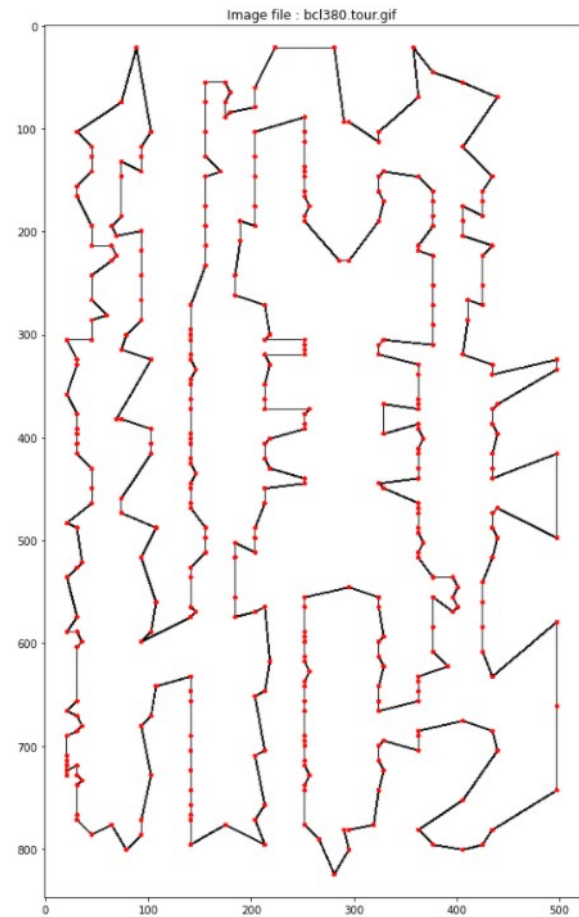


Fig. 24. Actual result [2]



## REFERENCES

- [1] Artificial Intelligence: a Modern Approach, Russell and Norvig (Fourth edition)
- [2] VLSI datasets-collection of 102 TSP instances provided by Andre Rohe  
<http://www.math.uwaterloo.ca/tsp/vlsi/index.html#XQF131>