Simulated Annealing

Travelling salesman problem using simulates annealing

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

Anand Mundhe
MTECH CSE (2020-22)
IIIT Vadodara
Gandhinagar, India
202061006@iiitvadodara.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

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import notebook
class coordinate:
    def __init__(self,x,y):
        self.x=x
        self.y=y

    @staticmethod
    def get_distance(a,b):
        return np.sqrt(np.abs(a.x-b.x)**2+np.abs(a.y-b.y)**2)

    @staticmethod
    def get_total_distance(coords):
        dist=0
        for first,second in zip(coords[:-1],coords[1:]):
            dist+=coordinate.get_distance(first,second)
        dist+=coordinate.get_distance(coords[0],coords[-1])
        return dist

def readTSP(f):
    f1=open(f)
    s=""
    flag=False
    coords=[]
    s=f1.readline()
    while s!="EOF\n":
        if s=="NODE_COORD_SECTION\n":flag=True
        elif flag:
            s=s.split()
            coords.append(coordinate(int(s[1]),int(s[2])))
        s=f1.readline()
    return coords

coords=readTSP("rajds.tsp")
f=open("sortedLocation")
s=" "
s=f.readline()
while(s!=""):
    print(s,end="")
    s=f.readline()

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
plt.figure(figsize=(10,10))
plt.title("100px is 50KM")
img = mpimg.imread('map RAJ.png')
imgplot = plt.imshow(img)
plt.show()

def driver(f,flag=False):
    #read and plot initial coordinates
    print("Working with :",f)
    coords=readTSP(f)
    plt.figure(figsize=(10,10))
    plt.title("Before TSP")
    for first,second in zip(coords[:-1],coords[1:]):
        plt.plot([first.x,second.x],[first.y,second.y],'b')
    plt.plot([coords[0].x,coords[-1].x],[coords[0].y,coords[-1].y],'b')
    for c in coords:
        plt.plot(c.x,c.y,'ro')
    plt.grid(True)
    print("Cost is : ",coordinate.get_total_distance(coords))
    if flag:plt.gca().invert_yaxis()
    plt.show()
    #Apply TSP
    from sys import stdout
    from time import sleep
    cost0=coordinate.get_total_distance(coords)
    T=30
    factor=0.999
    T_init=T
    epochs=10000
    for i in notebook.tqdm(range(epochs),total=epochs,unit="epoch"):
        stdout.write("\rNew cost is %f" % cost0)
        stdout.flush()
        #sleep(0.5)
```

```
        T=T*factor
        for j in range(100):
            r1,r2=np.random.randint(0,len(coords),
    size=2)
            temp=coords[r1]
            coords[r1]=coords[r2]
            coords[r2]=temp
            cost1=coordinate.get_total_distance(
    coords)
            if cost1<cost0:
                cost0=cost1
            else:
                x=np.random.uniform()
                if x<np.exp((cost0-cost1)/T):
                    cost0=cost1
                else:
                    temp=coords[r1]
                    coords[r1]=coords[r2]
                    coords[r2]=temp
    #Plot obtained coordinates
    plt.figure(figsize=(10,10))
    plt.title("After TSP")
    for first,second in zip(coords[:-1],coords[1:]):
        plt.plot([first.x,second.x],[first.y,second.
    y],'b')
    plt.plot([coords[0].x,coords[-1].x],[coords[0].y
    ,coords[-1].y],'b')
    for c in coords:
        plt.plot(c.x,c.y,'ro')
    plt.grid(True)
    if flag:plt.gca().invert_yaxis()
    plt.show()

driver("rajds.tsp",True)
```

Listing 1. TSP using simulated annealing in Rajsthan

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

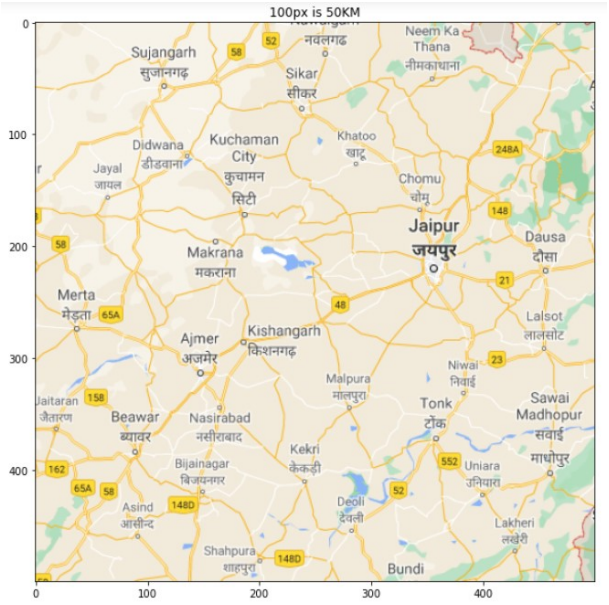

Fig. 2. Cities in Rajasthan on map

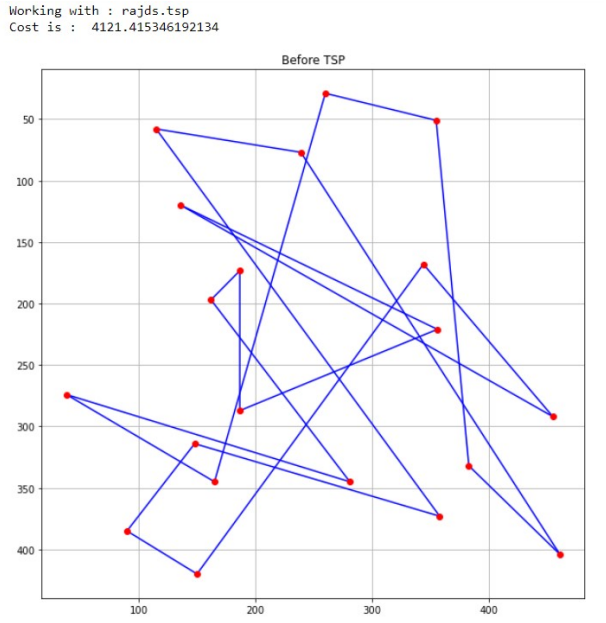

Fig. 3. Before applying TSP

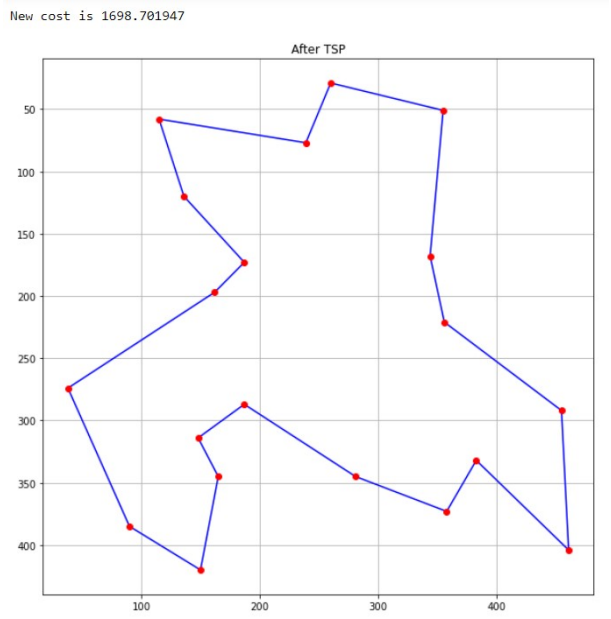

Fig. 4. After applying TSP

III. Solution of TSP of VLSI datasets

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
def image(f):
    plt.figure(figsize=(15,15))
    plt.title("Image file : "+f)
    img = mpimg.imread(f)
    imgplot = plt.imshow(img)
    plt.show()
def pimage(f):
    image(f+".points.gif")
def qimage(f):
    image(f+".tour.gif")

file1="xqf131"
file2="xqg237"
file3="pma343"
file4="pka379"
file5="bcl380"
files=[file1,file2,file3,file4,file5]
```

Listing 2. Setup for applying TSP on VLSI datasets

1) VLSI dataset 1

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

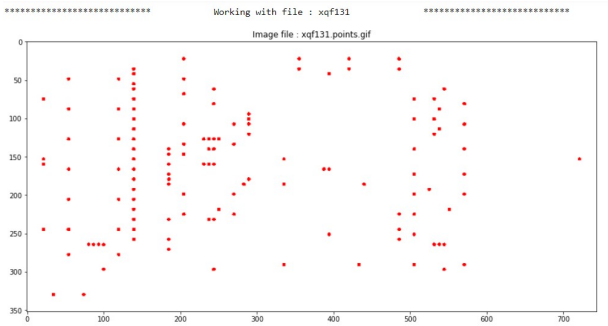

Fig. 5. VLSI dataset 1 [2]

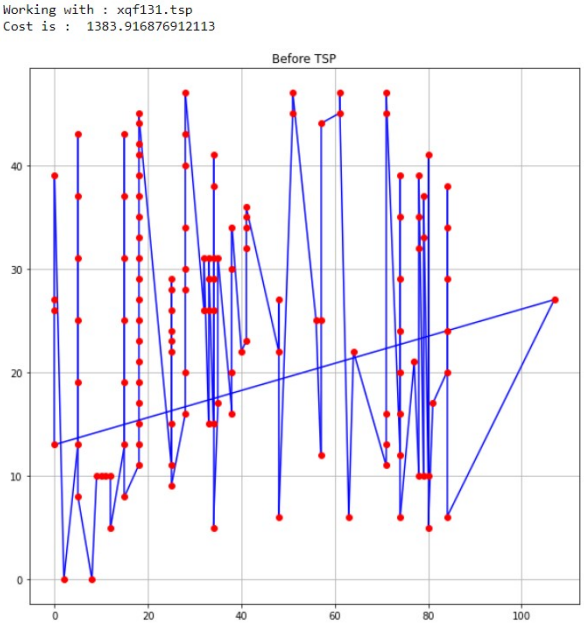

Fig. 6. Before applying TSP

```
New cost is 691.8130670
```

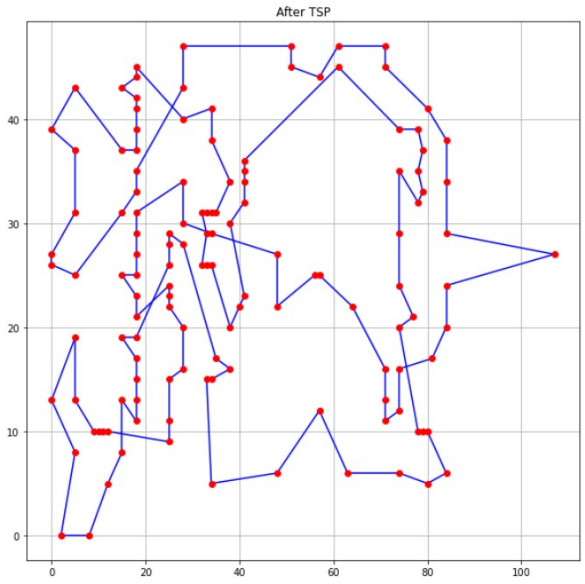

Fig. 7. After applying TSP

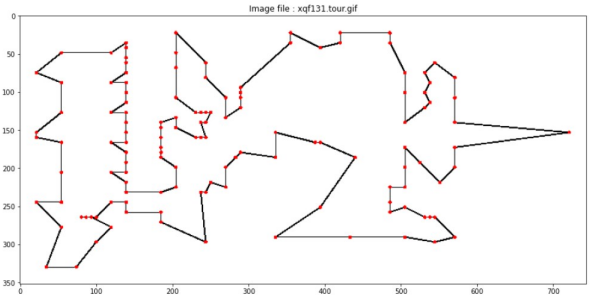

Fig. 8. Actual result [2]

2) VLSI dataset 2

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

```

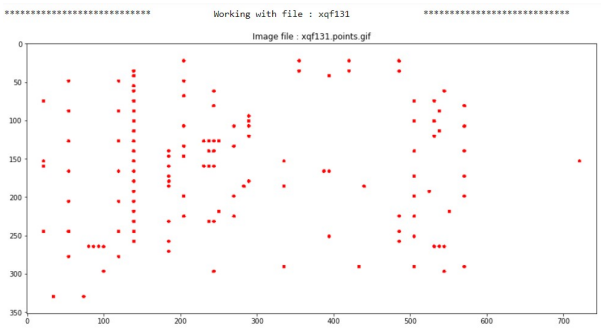

Fig. 9. VLSI dataset 2 [2]

```
Working with : xqg237.tsp
Cost is :  2949.5790533940426
```

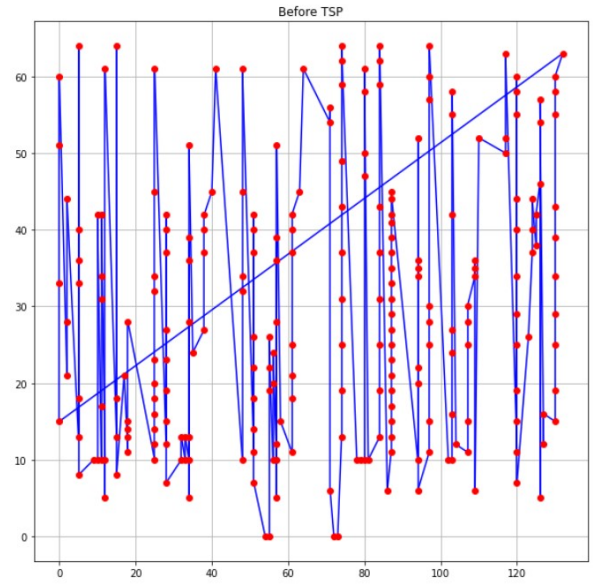

Fig. 10. Before applying TSP

New cost is 1667.093411

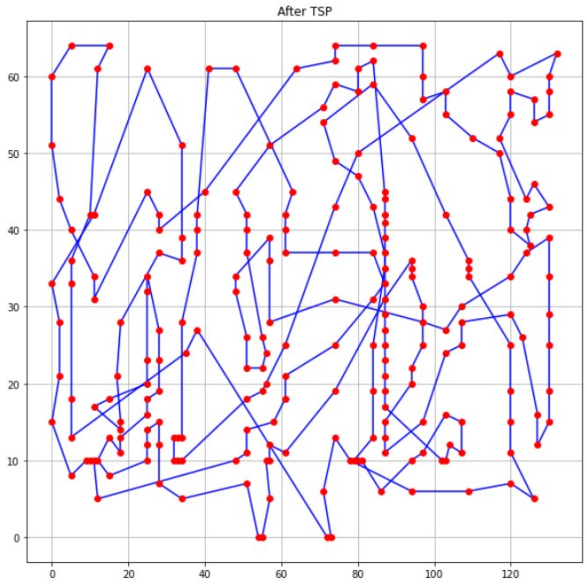

Fig. 11. After applying TSP

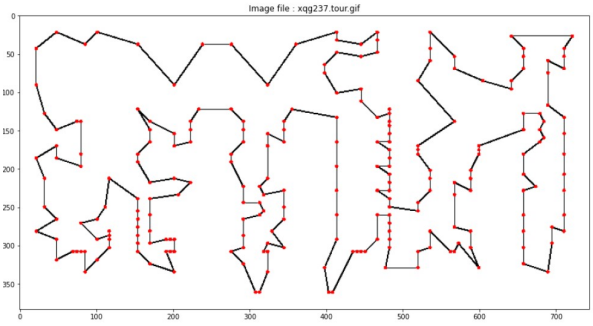

Fig. 12. Actual result [2]

3) VLSI dataset 3

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

```

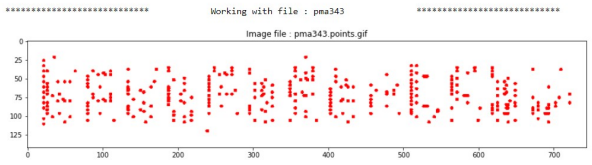

Fig. 13. VLSI dataset 3 [2]

```
Working with : pma343.tsp
Cost is :  3117.1798732795
```

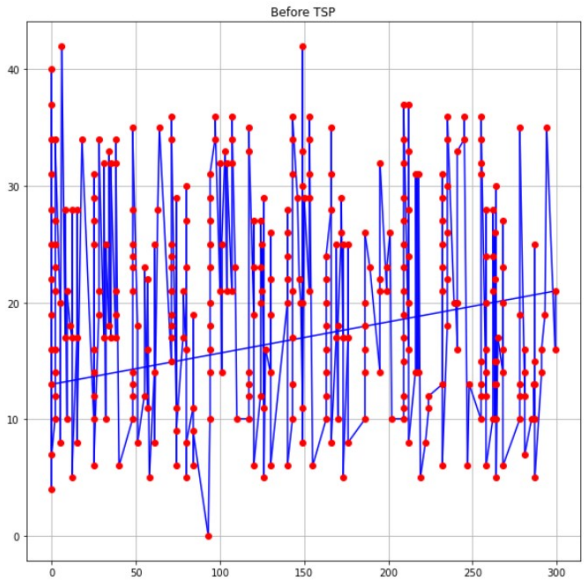

Fig. 14. Before applying TSP

```
New cost is 2167.790007
```

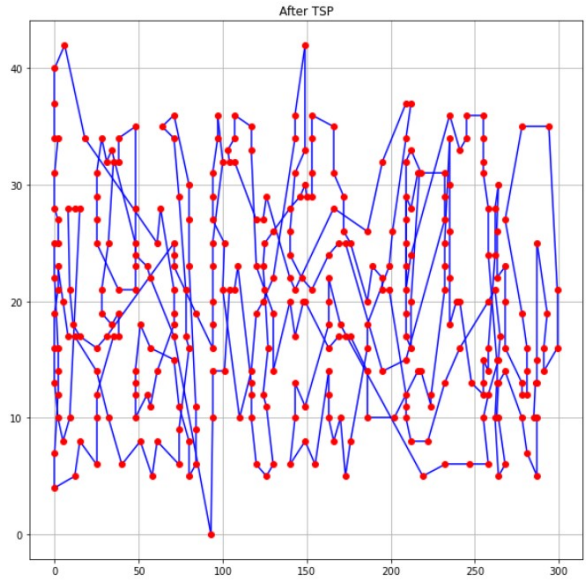

Fig. 15. After applying TSP

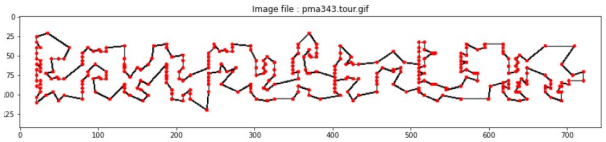

Fig. 16. Actual result [2]

4) VLSI dataset 4

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

```

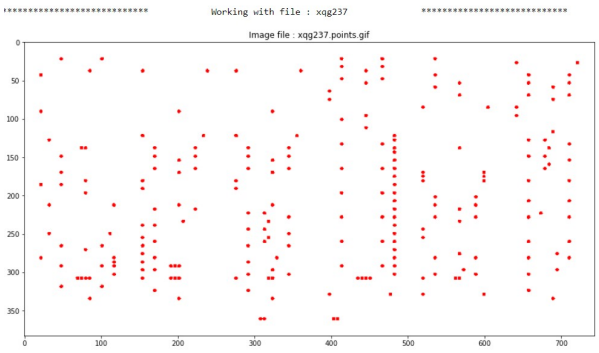

Fig. 17. VLSI dataset 4 [2]

```
Working with : xqg237.tsp
Cost is :  2949.5790533940426
```

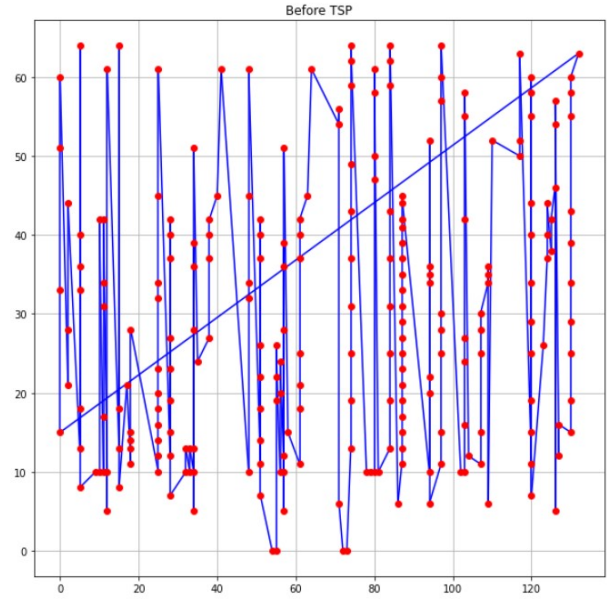

Fig. 18. Before applying TSP

```
New cost is 1667.093411
```

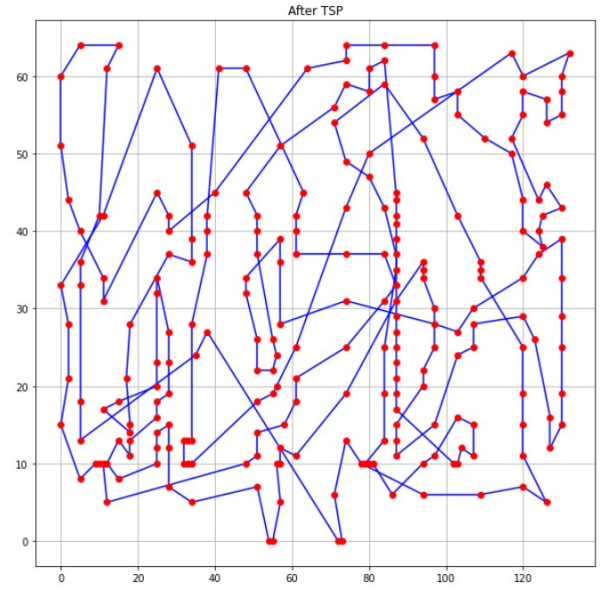

Fig. 19. After applying TSP

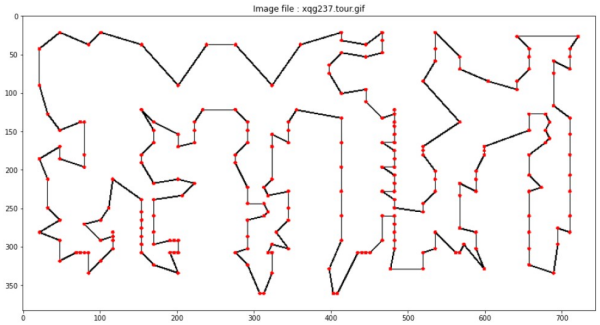

Fig. 20. Actual result [2]

5) VLSI dataset 5

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

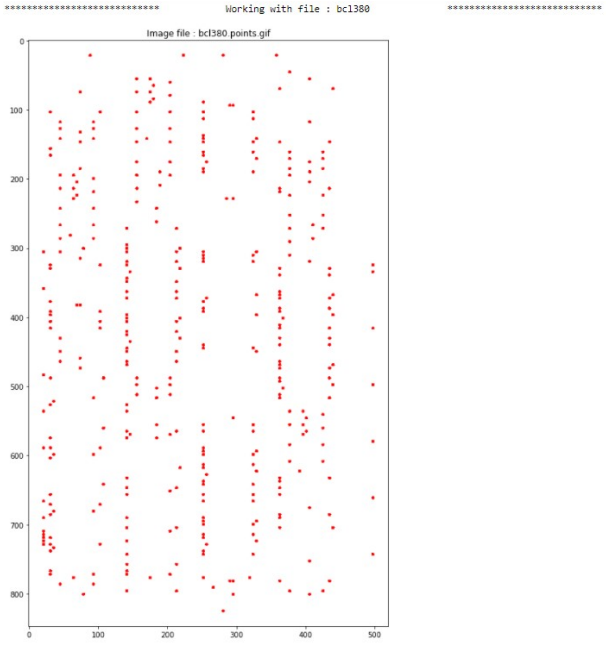

Fig. 21. VLSI dataset 5 [2]

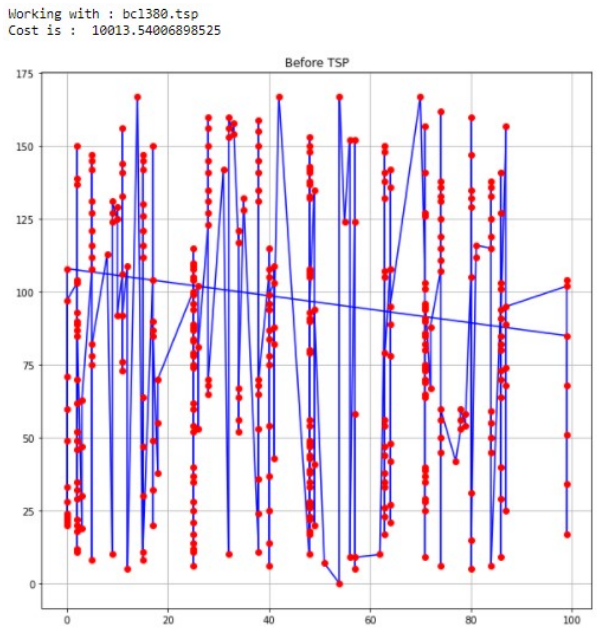

Fig. 22. Before applying TSP

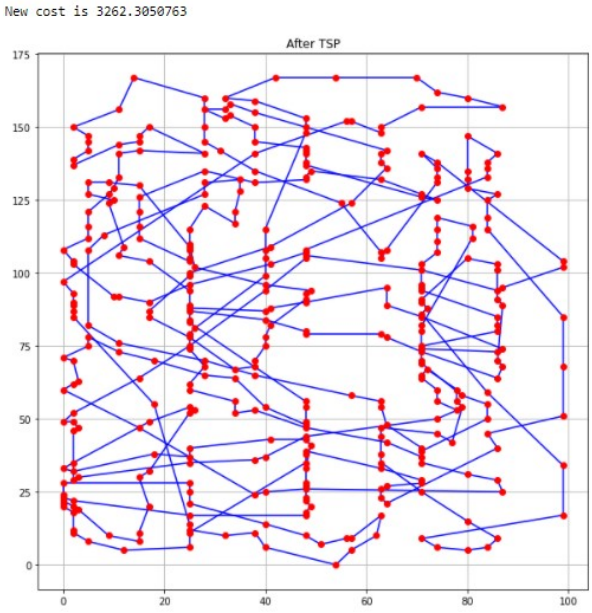

Fig. 23. After applying TSP

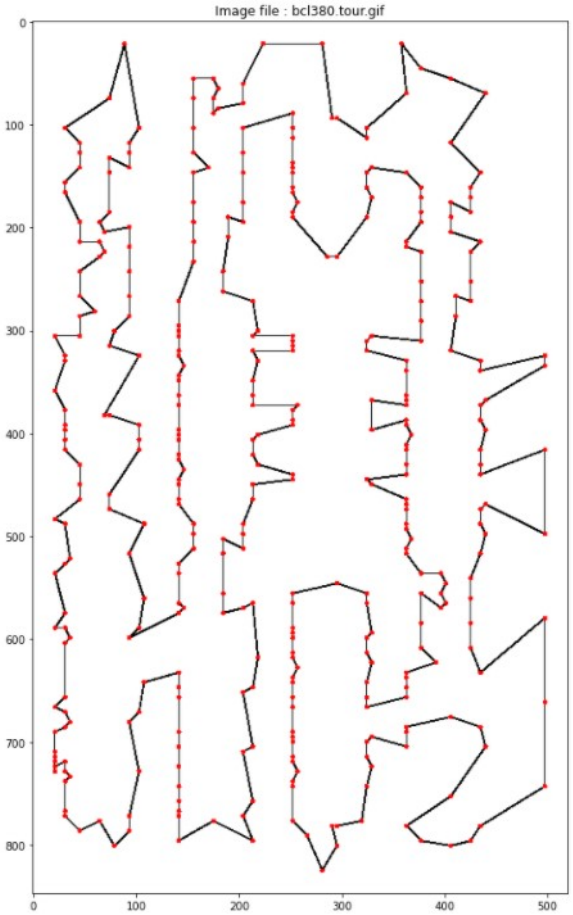

Fig. 24. Actual result [2]

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