Mathematics Club DC Project: Optimization

The Travelling Salesman Problem

Shivanshu

Sameer

Deenabandhan

April 2024

Abstract

In this work we try to solve the very famous Travelling Salesman Problem - "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

Here we solve an equivalent version of the problem in which we are given the coordinates of the cities in an X-Y system. The most naïve approach to the TSP would be to check every possibility; which would mean we'll have to check (n-1)!/2 different paths. We realize that the number of steps would shoot up even for numbers as small as 20. In fact, the TSP is an NP-hard problem, as showed by Richard M. Karp in 1972, so it doesn't have a polynomial-time solution. The problem is of interest to mathematicians because of its strong relation with graphs and Hamiltonian tours. It has wide-ranging applications from DNA sequencing to GPS navigation.

We try to solve the problem for a set of 40 cities using two approximate algorithms:

1 Simulated Annealing

Perhaps the most famous of all heuristic solutions is the Simulated Annealing, which as the name suggests, simulates the process of Annealing, but in a mathematical way.
Annealing is a metallurgical process involving the controlled cooling of a heated metal to alter its physical attributes. Similarly, in SA, we use a temperature-dependent probabilistic function to explore various paths. We start with a random path and start swapping cities randomly. If the length of our path decreases, we accept the swap; if not, we still accept it, but with a certain temperature-dependent probability given by:

$$p(\Delta E) = e^{-s\Delta E/T}$$

The idea is to explore the sample space first and then start becoming more selective as the algorithm proceeds. We start with a temperature T and with each iteration in the process, we keep reducing the temperature by a factor ($\gamma < 1$, but close to it). We stop iterating when the temperature reaches 1. So the complexity of our algorithm is O(logT/logγ). The factor "s" is the stochasticity, an increase of its value increases the selectivity of our algorithm. This gives an approximation for the shortest path. SA turns out to be a very good approximation for practical purposes. Also, in our code, we kept a track of minimum distance throughout, so the solution given by the code need not always be the one that comes at the end of SA.

Algorithm 1 Simulated Annealing

$T \leftarrow T_0$
$s \leftarrow s_0$
$\gamma \leftarrow \gamma_0 < 1$
$L \leftarrow L_0$ ▷ L is path length
while $T > 1$ **do**
choose $0 < n_1, n_2 <$ no. of cities + 1
swap the position of cities with indices n_1, n_2
$L_1 \leftarrow$ new path length
if $L_1 < L$ **then**
$L \leftarrow L_1$
else
with probability p$(L_1 - L)$:
$L \leftarrow L_1$

with probability 1 - p$(L_1 - L)$: swap the position of cities n_1, n_2
end if
$T \leftarrow \gamma T$
end while

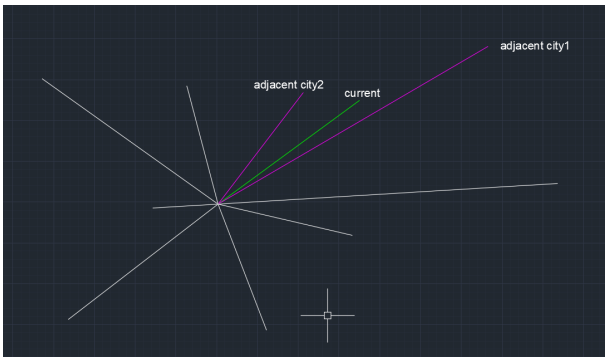


Figure 1: motivation behind polar optimisation

2 Polar optimisation

In this method, we depart from the naive greedy of starting at a city and then going to the nearest city and rather use minimum angular distance. In other words, we start at a city that is closest to the centroid, and then this as the starting point, we look at the two nearest cities that make the least angle with this city among all the cities.
Among these two we choose the city with the minimum angle. We now connect this with the original city and delete it. We repeat the procedure with the chosen city till only one city is left.
To ensure a closed path we connect the starting city with this last remaining city.This gives a good path to start with. The motivation behind this procedure is explained in Figure 1.
This clearly needs some improvement(Figure 3) and so we introduce 2 different heuristical improvements.

1. We go through the entire path once, take a pair of cities, and swap their neighbors if the distance decreases. More formally a "connectivity" exists between a pair (a,b) and (c,d) if a is connected with b and c, and b is connected with a and d. By this swap operation, we establish a connectivity between (b, a) and (c,d).

2. We again go through the entire path once, take 3 cities and shift the middle city between the two right(or left) neighbors of this trio. More formally if cities a,b,c,d,e are connected in that order then we either connect b to d and e, disconnect d and e and then connect a and c or, connect d to a and b, disconnect a and b and connect c and d, if the distance decreases.

These improvements reduce cross-connections or make the path more "spread-out". We apply these steps one after the other, twice, i,e., 1 →2 →1 →2.
The overall time complexity of polar optimization alone is O(nlogn), where n is the number of cities. This is because the sorting of angles is O(nlogn), initial order generation is O(n) and both the improvements are also O(n).

The entire process is summarised as Algorithm 2.

Algorithm 2 Polar optimisation

current ← argmin(distance from centre)
$L \leftarrow 0$
iteration ← 0
while iteration < no.of cities - 1 **do**

$\theta_1 \leftarrow$ min(angle between current and any city clockwise)
$\theta_2 \leftarrow$ min(angle between current and any city anticlockwise)
previous ← current
current ← argmin(θ_1, θ_2)
$L \leftarrow L$ + distance(previous,current)
end while
$L \leftarrow L$+distance(start city,last city)

iteration ← 0
a,b← two connected cities
c,d← cities connected to a,b resp
while iteration < no. of cities **do**
if distance(a,c) + distance(b,d) > distance(b,c) + distance(a,d) **then** swap(a,b) , update L
end if
a,b← b,c
end while

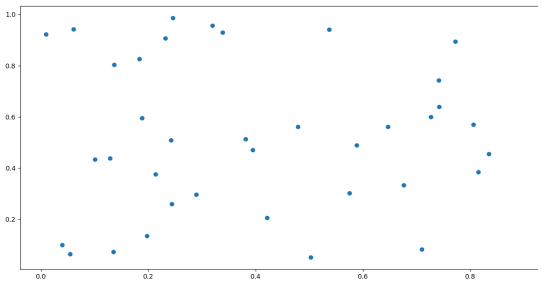
iteration ← 0
a,b,c,d,e ← five connected cities
if distance(a,b) + distance(b,c) + distance(d,e) > distance(a,c) + distance(b,d) + distance(b,e) **then** swap(b,c) , swap(c,d) , update L
else
if distance(a,b) + distance(c,d) + distance(d,e) > distance(a,d) + distance(b,d) + distance(c,e) **then** swap(c,d) , swap(b,c) , update L
end if
end if

3 Convex Hull Method

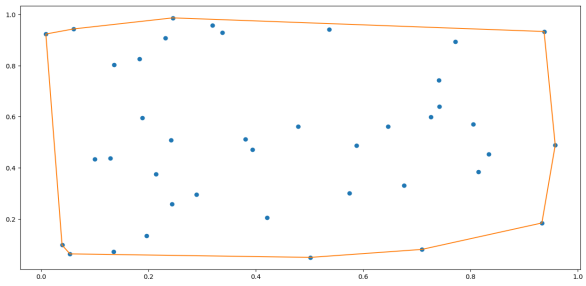
This method is just an intuitive method which we came up with. It involves a usage of a geometric concept called Convex Hull.

Definition : In geometry, the convex hull, convex envelope or convex closure of a shape is the smallest convex set that contains it.

For example for the given set of points , the convex hull shape would be

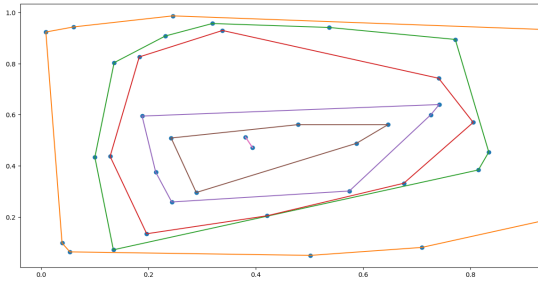


(a) The set of points

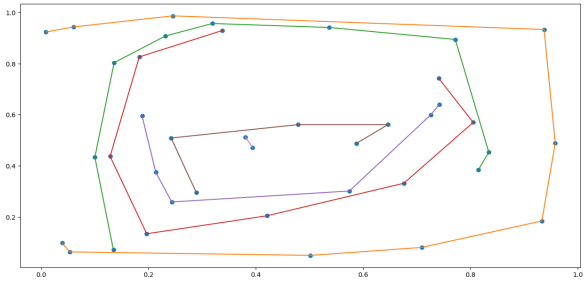


(b) The convex hull shape

Let's form a series of convex hull shapes and disconnect the largest path formed in between the points of the convex hull shapes



(a) The set of convex hull shapes



(b) The set of convex hull shapes with disconnectivity

As a final step connect the convex hull shapes to the nearest available points of the convex hull shapes

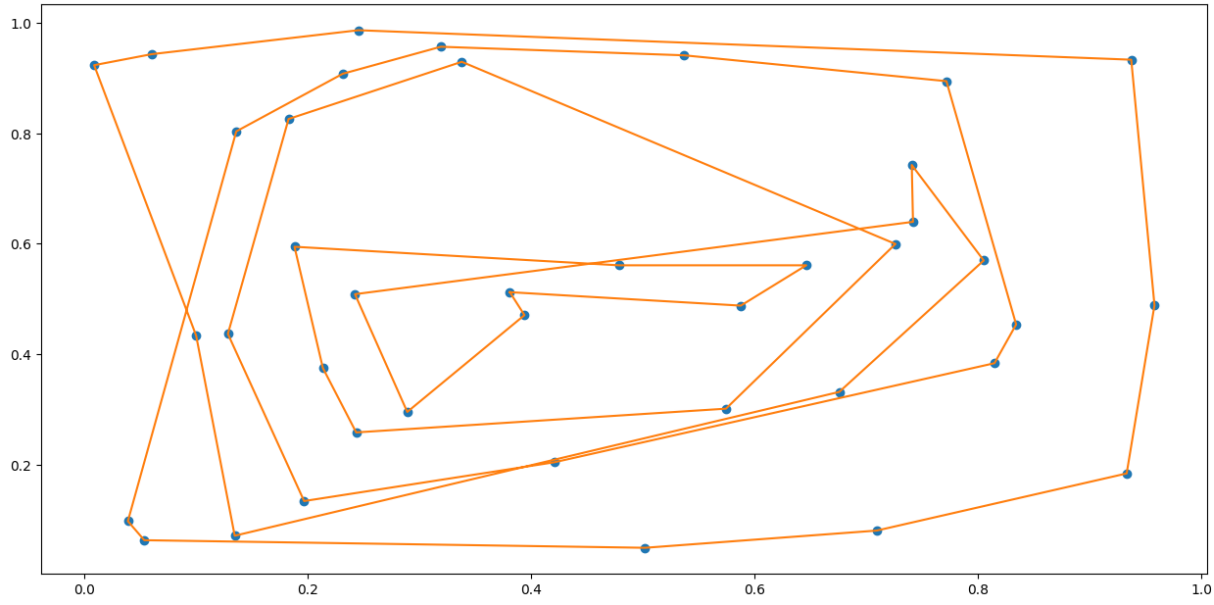


Figure 4: The final path that we would get after performing all operations

Algorithm 3 Convex Hull Method

$arr \leftarrow$ Set of points
$N \leftarrow$ length of arr[]
$path \leftarrow$ the final array storing the order of points
$path \leftarrow []$
while $N > 0$ **do**
Create a convex Hull shape for the set arr Store it in a temporary array tmp[]
    $n \leftarrow$ no. of cities in tmp
    i $\leftarrow$ 0
    mx $\leftarrow$ distance(tmp[0],tmp[n-1])
    p $\leftarrow$ the value the contains the second element of the two that has the largest path
    p $\leftarrow$ 0
    while i < n **do**
        if distance(tmp[i],tmp[i+1]) > mx) **then**
            mx $\leftarrow$ distance(tmp[i],tmp[i+1])
            p $\leftarrow$ i+1
        end if
    end while
    tmp[ ] $\leftarrow$ an array with starting element at p such that starting and ending elements i.e tmp[p] , tmp[p+1] are disconnected
    $n_a \leftarrow$ the number of elements in path[ ]
    i $\leftarrow$ 0
    mx $\leftarrow$ distance(path[0],tmp[n-1])+distance(path[n-1],tmp[0])
    p $\leftarrow$ the value the contains the position of the element that is to be connected with the convex hull
    p $\leftarrow$ 0
    while $i < n_a$ **do**
        if distance(path[0],tmp[i])+distance(path[n-1],tmp[i+1]) > mx **then**
            mx $\leftarrow$ distance(path[0],tmp[i])+distance(path[n-1],tmp[i+1])
            p $\leftarrow$ i+1
        end if
    end while
    tmp[ ] is appended to the pth position of the path[ ] array
end while
path[ ] contains the set of points that are given in the order in which the closed path occurs

Results

1. Only Simulated Annealing
   Using just simulated annealing we obtained the plots in Figure 2 using different values of s and gamma.

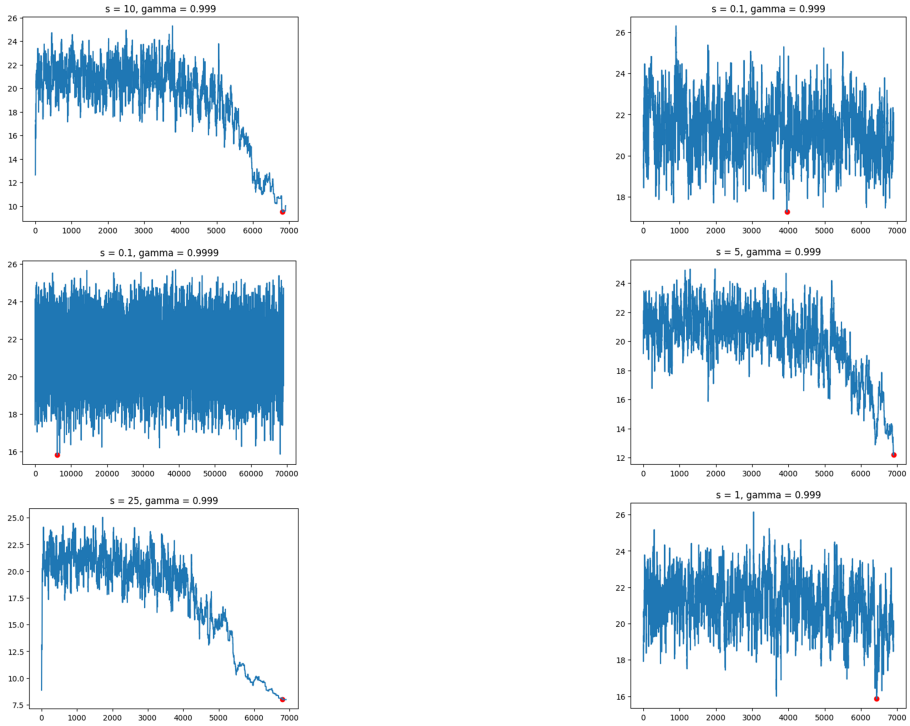


Figure 5: plots of simulated annealing

This indicates that gamma is responsible for the number of iterations and s is responsible for flexibility in exploration as the distance increases in the initial iterations.

We also note that for both s and γ there exists an optimal value, without a monotonic trend. For our setting optimal s = 25 , $\gamma = 0.999$
The average distance found by performing SA 100 times was 7.257 with average run time of 0.393s . The path obtained in one of the iterations is shown in Figure 6

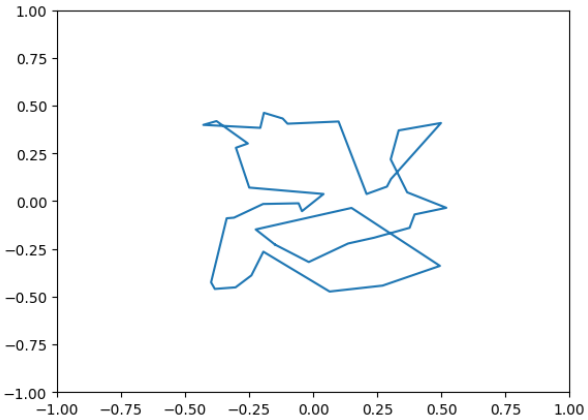
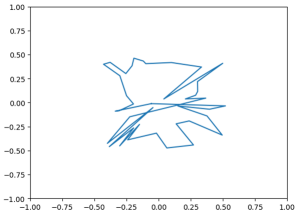


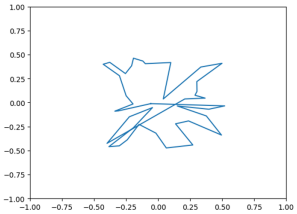
Figure 6: using just SA

2. Polar optimization
   The initial order obtained by step 1 of Algorithm 2 is shown in Figure. Subsequent applications of step 2 and step 3 (2 times) generates paths shown in Figure 7
   The path lengths obtained were about 9.25, 8.13 , 7.63, 7.52, 7.07 in five steps. Not much change was seen on applying step 2 and 3 after this.
   We attempted introducing randomness in step 2 and 3 by introducing a probability func-

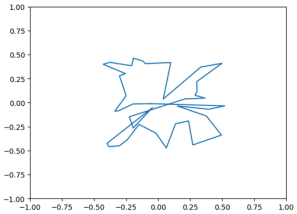
tion $p(\Delta E) = e^{-s\Delta E/T}$ but it rather increased the overall path. Probably, this is because of a sharper temperature decay and also the small sample space used.



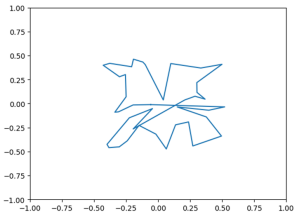
(a) step1



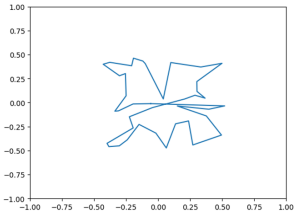
(b) step2



(c) step3



(d) step4



(e) step5

Figure 7: plots of polar optimization

3. Convex hull method
   This method involves mere connection of convex hull shapes,it turns out to optimise in the maxima sense, giving distance of about 11.39

4. Comparison with a popular TSP solver: Ref - 1 reveals that the optimal path length is about 5.40.

Conclusion

Thus we have implemented an approximate solution to the Travelling Salesman Problem. The best distance given by SA is about 6.38 though on average we get 7.25. Our algorithm gives a distance of 7.07 outperforming SA and is a lot faster. The error with respect to a well known tsp-solver is about 31%

References

Some of the notable references we used were:

1. https://www.lancaster.ac.uk/fas/psych/software/TSP/TSP.html
2. https://www.youtube.com/watch?v=GiDsjIBOVoA
3. https://www.geeksforgeeks.org/convex-hull-algorithm/
4. Combinatorial optimization - springer by Korte and Vygen