Travelling Salesman Problem

19UCS011	**Amit Gupta**
19UCS042	**Vipul Aggarwal**
19UCS045	**Rishiraj Yadav**
19UCS049	**Shashank Bansal**
19UCS122	**Tanay Makharia**

Introduction

The travelling salesperson problem (TSP) is a classic optimization problem where the goal is to determine the shortest tour of a collection of *n* cities (i.e. nodes), starting and ending in the same city and visiting all of the other cities exactly once.
In this project we are going to use A* search strategy to find the optimal solution to the problem and use the Minimum Spanning Tree as the heuristic function.

Input File Format

N
d_{11} d_{12} d_{13} ... d_{1N}
d_{21} d_{22} d_{23} ... d_{2N}
.....
.....
d_{N1} d_{N2} d_{N3} ... d_{NN}

where,
N = number of cities
City1 ... N is the cities. (1 is the initial city)
d_{ij} = distance between $City_i$ and $City_j$.

$d_{ij} = d_{ji}$ and all the distances are integers.

Output File Format

City1->City2->City3->... ->CityN->City1
Minimum Distance= d (computed minimum distance)

Travelling Salesman Problem

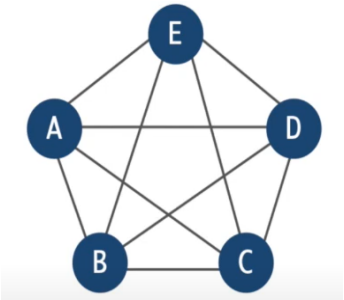
using MST Heuristic

Our implementation of an MST heuristic involves four main steps.

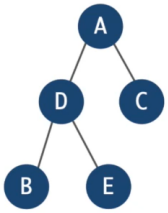
- First, every edge in the graph is placed in a priority queue, giving priority to the lowest-costing edge.
- Second, we use Prim's algorithm to create a minimum spanning tree. This is done by arbitrarily picking a start node and using the priority queue to repeatedly add the current lowest cost edge connected to the tree, one edge at a time, until all vertices are included.
- Third, this minimum spanning tree is transformed into an eulerian tour by DFS traversal.
- Fourth, the eulerian tour is transformed into an Asymmetric TSP solution by traversing its edges, replacing each edge that would visit an already visited node with an edge to the next node along the eulerian tour that hasn't yet been visited.

This results in an Asymmetric TSP solution because each node will be visited only once, and it will end at the start node because that is what the eulerian tour already does. To ensure a solution is found in incomplete graphs, our algorithm repeats steps 2 through 4 with each city as a starting node for the MST and returns the best solution found.

Taking an example of a graph of 5 cities,



Suppose the MST of the above graph (converted by using Prim's Algorithm) is:-



Now apply DFS on the above spanning tree, the result will be:-
A->D->B->D->E->D->A->C->A

Here, there's a duplicacy of the cities. To remove the duplicates, we can apply the heuristic of the triangular inequality which is:-
The sum of the length of the two sides of a triangle is greater than the length of the third side, i.e., in the above example, d(B,E)<d(B,D)+d(D,E). Thus the shorter path BE should be chosen in order to get the minimum possible distance.
Similarly, d(E,C)<d(E,D)+d(D,A)+d(A,C) so the path EC should be chosen.
Hence the resulted distance will be:-
A->D->B->E->C->A

Our resulted tour will lie between
Most Optimal Result ≤ Resulted Tour ≤ 2*MST
1) The cost of the best possible Travelling Salesman tour is never less than the cost of MST.

2) The total cost of full walk is at most twice the cost of MST (Every edge of MST is visited at-most twice)

3) The output of the above algorithm is less than the cost of full walk. In the above algorithm, we print preorder walk as output. In preorder walk, two or more edges of full walk are replaced with a single edge. So if the graph follows triangle inequality, then this is always true.

We can conclude that the cost of output produced by the approximate algorithm is never more than twice the cost of the best possible solution.

Pros and Cons of MST Heuristic:-

Pros:

- The main strength of this algorithm is its potential to find a solution close to the optimal.
- This algorithm will be faster because implementation of prim's algorithm will be to $n^2 \log(n^2)$ (using priority queue).Overall time complexity will be $O(n^2 \log(n))$.

Cons:

- The cons of this algorithm lie in the shortcutting of the eulerian tour. In order to transform the eulerian tour into an Asymmetric TSP solution, we must remove the edges that lead to already-visited nodes, while still covering each node once.
- For a TSP problem that satisfies the triangle inequality, shortcutting is very effective because it replaces a path of several edges between nodes with a single edge, which is always shorter by the definition of triangle inequality. But all TSP problems do not satisfy the triangular inequality.

Therefore, this shortcutting method becomes inefficient, and results in worst-case result.

Travelling Salesman Problem

using A* Graph Search Algorithm

A* algorithm works based on heuristic methods and this helps achieve optimality. A* is a different form of the best-first algorithm. It also offers completeness, if there is any solution possible to an existing problem, the algorithm will definitely find it.
When A* enters into a problem, firstly it calculates the cost to travel to the neighbouring nodes and chooses the node with the lowest cost. If f(n) denotes the cost, A* chooses the node with the lowest f(n) value. Here 'n' denotes the neighbouring nodes.
The calculation of the value can be done as:
$f(n)=g(n)+h(n)$
where,
g(n) = shows the shortest path's value from the starting node to node n
h(n) = The heuristic approximation of the value of the node

Here h(n)= distance to the nearest unvisited city from the current city + estimated distance to travel all the unvisited cities(MST heuristic used here) + nearest distance from an unvisited city to the start city.

Here we have underestimated the heuristic function to make the algorithm admissible so that we can get the most optimal result.

Pros and Cons of A* Search:-

Pros :

- It is an informed search and optimal approach. It relies on an open list as well as a closed list to find a path that is optimal and complete towards the goal.

Cons:

- Uses a lot of memory since each node created has to be kept accounted for. From the dataset, it can be verified that when there is a greater number of cities, the algorithm will take a considerably larger amount of time.

WHY A* Heuristic Search is better than MST heuristic?

A* gives more optimal result because in A* search algorithm we apply MST at each level.

Our Findings -

Attached results shows that A* is significantly giving more optimal results than Minimum Spanning Tree(MST) heuristic.

INPUT 1 -

N = 5
0 3 4 2 7
3 0 4 6 3
4 4 0 5 8
2 6 5 0 6
7 3 8 6 0

Using MST heuristics -

```
0 -> 4 -> 2 -> 1 -> 3 -> 0
Minimum distance with MST :  71
```

Using MST heuristics and A*-

```
0 -> 2 -> 1 -> 3 -> 4 -> 0
Minimum distance :  64
```

INPUT 2-
N = 5
0 15 13 16 12
15 0 14 11 17
13 14 0 19 18
16 11 19 0 14
12 17 18 14 0

Using MST heuristics -

```
0 -> 3 -> 1 -> 4 -> 2 -> 0
Minimum distance with MST :  23.0
```

Using MST heuristics and A* -

```
0 -> 2 -> 1 -> 4 -> 3 -> 0
Minimum distance :  19.0
```

INPUT 3-
N = 15

0 29 82 46 68 52 72 42 51 55 29 74 23 72 46
29 0 55 46 42 43 43 23 23 31 41 51 11 52 21
82 55 0 68 46 55 23 43 41 29 79 21 64 31 51
46 46 68 0 82 15 72 31 62 42 21 51 51 43 64
68 42 46 82 0 74 23 52 21 46 82 58 46 65 23
52 43 55 15 74 0 61 23 55 31 33 37 51 29 59
72 43 23 72 23 61 0 42 23 31 77 37 51 46 33
42 23 43 31 52 23 42 0 33 15 37 33 33 31 37
51 23 41 62 21 55 23 33 0 29 62 46 29 51 11
55 31 29 42 46 31 31 15 29 0 51 21 41 23 37
29 41 79 21 82 33 77 37 62 51 0 65 42 59 61
74 51 21 51 58 37 37 33 46 21 65 0 61 11 55
23 11 64 51 46 51 51 33 29 41 42 61 0 62 23
72 52 31 43 65 29 46 31 51 23 59 11 62 0 59
46 21 51 64 23 59 33 37 11 37 61 55 23 59 0

Using MST heuristics -

```
0 -> 12 -> 1 -> 14 -> 8 -> 4 -> 7 -> 9 -> 11 -> 13 -> 2 -> 6 -> 5 -> 3 -> 10 -> 0
Minimum distance with MST :  366.0
```

Using MST heuristics and A* -

```
0 -> 10 -> 3 -> 5 -> 7 -> 9 -> 13 -> 11 -> 2 -> 6 -> 4 -> 8 -> 14 -> 1 -> 12 -> 0
Minimum distance :  291
```

Project Link -
https://github.com/amitgupta20/AI-Project-TSP
(implemented using Python)
Files Attached - Data Set , Output, Code , Report.

References

Prim's Algorithm:-
https://www.tutorialspoint.com/data_structures_algorithms/prims_spanning_tree_algorithm.htm

A* Heuristic Search:-
https://www.geeksforgeeks.org/a-search-algorithm/

Dataset Used:-
https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html

GitHub link -
https://github.com/amitgupta20/AI-Project-TSP