Maze Simulation

Prakhar Aggarwal, Dhairya Gupta

July 29, 2021

1 Travelling Salesman Problem

We are given a list of cities and the distances between each pair of cities. Our goal is to find the shortest possible route that visits each city exactly once and returns to the origin city covering the least distance. The Travelling Salesman Problem is NP-complete i.e. there is no known polynomial time solution for this problem. Note that Minimum Spanning Tree problem and Travelling Salesman Problem are two different problem.

1.1 Task

We are given an weighted undirected Graph G = (V,E). Every edge (u,v) ϵ E is assigned a non-negative cost c(u,v). We have to find a **hamiltonian cycle** (A path that visits each vertex exactly once and there is an edge from the last vertex to the starting vertex). We will define c(A) as the total cost of the edges in A where A $\subseteq$ E.

1.2 Representation

We can represent the graph as an adjacency matrix. The (i,j) element of this matrix denotes the c(u,v). c(u,u) is considered to be 0 i.e. there is no self-loop.

2 Approach

As Travelling Salesman Problem is NP-complete, there exists no polynomial time algorithm. We will look at some algorithms which give near optimum solutions with.

2.1 Brute Force Method

In brute force approach, we will explore all possible cycles which connect all nodes, calculate each cycle's cost and take the one with minimum cost.

Runtime Analysis: Time complexity for this approach = Total Number of Cycles * Time taken to calculate cycle's cost. As our source node is fixed, we have remaining n-1 nodes. As the graph is a complete graph, each node has n-1 edges. So source node has (n-1) options, selected node will have (n-2) options and so on. So, total number of cycles = (n-1)*(n-2)*......*2*1 i.e. Total Number of Cycles = (n-1)!. Assuming that we calculate each cycle's cost as we traverse over it, we can find a cycle's cost in O(1) time.

Hence, **Time Complexity = O(n!)**. This solution is not feasible enough to work on real life examples.

2.2 Approximated Method (Minimum Spanning Tree)

This method uses the Minimum Spanning Tree algorithm and the triangle inequality of the cost function (**c(u,w) $\leq$ c(u,v) + c(v,w)**) Using the approximated method, we can find a lower bound to the optimum solution.

Algorithm

- Let r be a node ϵ G(V), we will call it as the root vertex.
- We will compute the Minimum Spanning Tree with r as the root. (We will use **Prim's algorithm** for computing the Minimum Spanning Tree)
- Let H be the list of vertices occuring in the order they are visited while doing a preorder traversal.
- H computed in the previous step will be the Hamiltonian Cycle

Graphical Visualization for the above steps:

Runtime Analysis:

- For computing Minimum Spanning Tree using prim's algorithm: $O(V^2)$ i.e. $O(n^2)$.
- Preorder traversal will take O(n) time (Number of vertices)

So, the overall time complexiy of this approach will be $O(n^2)$ which can be improved to O(ElogV) using adjacency list and binary heap. But, as the graph is complete, E will be of the order of n^2, hence, overall time complexity for Prim's algorithm will be $O(n^2)$.

Note that the above approach is just a lower bound to the optimal solution that too when the cost function satisfies the triangle inequality.

2.3 Dynamic Programming Method

This approach improves upon the factorial runtime in brute force method. In this method, we compute the Travelling Salesman Problem for ith vertex by using knowledge of previous vertices.

Algorithm

- Let r be a nodeG(V), we will call it as the root vertex.
- For every other vertex i (other than the root vertex), we find the minimum cost path with r as the starting point, i as the ending point and all vertices appearing exactly once.
- Let the cost of this path be dp(i), the cost of corresponding Cycle would be dp(i) + c(i, 1).
- Finally, we return the minimum of all dp(i) + c(i, r) values.
- Now, to compute dp(i), we will use the following recursive relation:
  - Let us define a term C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at root.
  - We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset.
  - We will take the minimum of all values.
  - Then we calculate C(S, i) for all subsets S of size 3 and so on.

Memoization in an array of size $N \times 2^N$ ensures each subproblem is solved exactly once.

Algorithm 1: dp(root, vis)

```
vis ← visit(vis, root);
ans ← INT_MAX;
forall neighbor of root do
    if !isVisited(vis, neighbor) then
        pathLength ← edgeLength(root, neighbor) + dp(neighbor, vis);
        ans ← min(ans, pathLength);
    end
end
return optimalLength;
```