DD2440 Advanced Algorithms
Travelling Salesperson 2D

Sashikanth Raavikanti	sraa@kth.se
Bharat Sharma	bsharma@kth.se
Achyuth Ajith	achyuth@kth.se

December 6, 2019

Kattis submission Id - 5081976

Target Grade - E

1 Introduction

1.1 Problem

Travelling sales person is a NP- hard problem in combinatorial optimization, which is important in various fields of computer science that states that if given a list of cities and the distances between each pair of cities, you need to find the shortest possible route that visits each city and returns to the origin city.

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. There are lots of algorithms and heuristics for this complex problem and still researchers are working on improving them.

TSP can be modelled as an un-directed weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once.

If there are N cities, each represented as a graph as mentioned earlier with vertexes (x_i, y_i), we need to find the shortest path in which we can travel all the N vertices. The distance between 2 points is calculated as the Euclidean distance between 2 points, rounding to the nearest integer.

$distance_{(x1,y1)(x2,y2)} = \sqrt{(x1-x2)^2 + (y1-y2)^2}$

2 Methodology

2.1 Implementation

We used Java for our implementation. For the implementation we created a data structure named Node that represents a city. This structure will have information of adjacent cities (nodes), the parent city / node (for MST), the corresponding city index and the number of edges (degree). This Node structure is used and manipulated in all our algorithm implementations.

As a first step, we read all the city coordinates and create a distance matrix that maintains information of the distance between any 2 cities. This avoids calculating distance multiple times for different algorithms.

We implemented Nearest Neighbour and approximation algorithms for finding the minimum path. As part of approximation algorithms, we implemented an approximation based on Minimum Spanning Tree and another

one is Christofides algorithm. We ran optimizations on all the minimum paths we got from the algorithms. We returned the one that is minimum after optimization.

The details of each of the algorithms we used are in subsequent sections.

3 Heuristics

3.1 Nearest neighbour

We implemented nearest neighbour(naive) where the nearest node to the current node is added into the path. The NN heuristic is a naive and greedy algorithm. It works by repeatedly adding the city closest to the city that was added to the tour most recently.

3.2 Local optimization

In case of TSP, we take an existing tour and make some changes on some parts of the tour such that the resulting tour is shorter than the previous one.

2-opt

We tried 2 optimization where we find 2 pair of adjacent cities $a \to b$ and $c \to d$ in the current tour. Then we make a swap such that the sum of distances between the arrangement of cities is less that the old one i.e $a \to c + b \to d < a \to b + c \to d$. 2 opt helped increase the score for our implementation.

2.5-opt and 3-opt

We tried 2.5 and 3 opt as well. 2.5 opt is where we have adjacent cities $a \to b \to c$ and $d \to e$. Then we move b from in-between a and c and put it in between d and e and check if $a \to c{+}d \to b{+}b \to e < a \to b{+}b \to c{+}d \to e$.

For 3 opt, we do similar to what we did for 2 opt but instead of using combination of 2 edges, we use 3.

Our score didn't improve by implementing both 2.5 and 3 opt and that might be because our implementation of these both optimizations might be wrong.

3.3 Christofides algoritm

Christofides' is a tour construction heuristic that can be applied to graphs with the following property: a complete graph $G = (V, E, w)$ with edge weights that satisfy the triangle inequality $w(x, y) + w(y, z) \leq$w(x,z).

Step 1: Find a minimum spanning tree T of G.

We used Prim's Algorithm for minimum spanning tree. The algorithm maintains an array Q with the vertices that are not yet in the tree (initially, Queue is empty). It iterates through each vertex v not yet in the tree (using vertex 0 as the intial vertex) and chooses the minimum weight edge (u, v) where u is already in the tree. Thus, this is the lightest edge crossing the cut (since it connects an edge in the MST with an edge not yet in the MST). Then vertex v is added to the tree. The algorithm continues until Q is empty and thus all vertices have been added to the Tree.

Step 2: Let O be the set of vertices with an odd degree in T.

Now we have to find the vertices with odd degree in the Minimum Spanning Tree. We made a class Node that has a class element called degree that determines the degree of the vertex.

Step 3: Find a minimum cost perfect matching M for these vertices.

We need to make extra edges for the odd vertices to make them connected, so we have to perfectly match them. We used the brute force method mentioned in the KTH website for minimum cost perfect matching.

Step 4: Add M to T to obtain multigraph H.

We can make the new multigraph by adding the set of matched vertices.

Step 5: Find a Eulerian tour of H.

Now we find a euler circuit starting at any arbitrary node in our multigraph otained above. If our node has neighbors, we push our node on a stack, choose a neighbor, remove the edge between them from the graph, and make that neighbor the current vertex. If our vertex has no neighbors left, we add it to our circuit and pop the top vertex from the stack to use as our current vertex. We continue tracing a tour in this manner until the stack is empty and the last vertex has no more neighbors left.

Step 6: Convert the Eulerian tour into Hamiltonian path by skipping visited nodes (using shortcuts).

Finally, we turn our Euler circuit into a Hamiltonian path by walking along

the Euler tour, checking at every stop whether that node has already been visited. If it has, we skip that node and move on to the next one. Since our graph satisfies the triangle inequality, shorting vertices in this manner will not increase the length of our path.

4 Analysis

We started our approach by implementing the Nearest Neighbour (NN) heuristic to check the minimum score. Later, we implemented a local search heuristic called Genetic Algorithm. We did not get an optimal score with it as we believe it got stuck at a local maximum. We changed our approach to implement another local search heuristic called Simulated Annealing. We did not get an optimal score with that as well which we believe is for the same reason. We changed our approach to approximation algorithms. We tried approximation algorithm using Minimum Spanning Tree (MST) and Christofides. Although our score improved we did not reach the minimum score that is needed. Our score improved a lot using optimizations. We got the required score after 2 optimization. We tried the implementation of 2H and 3 optimization as well but weren't successful. Let us analyze on the time complexity for each of the algorithms we used here.

For finding the edge length where in we had to compute the distance between the given 2 vertices, time complexity of $\mathcal{O}(n^2)$.

For finding the nearest neighbours, time complexity of $\mathcal{O}(n^2)$.

For finding the MST using Prim's Algorithm, time complexity of $\mathcal{O}((V + E)logV)$

For finding the edges for the graph using Depth First Search, time complexity of $\mathcal{O}(V + E)$.

4.1 How to improve

One mistake we made was that we concentrated on different algorithms and implementations and didn't focus much on optimization initially. Later, when we started looking into optimization, it was probably too late and we couldn't implement it correctly. We have learned that improving the score is based on optimizations rather than different algorithms and we should have focused more on that. Therefore, implementing optimizations correctly and efficiently can help improve the score.

5 References

J. L. Bentley, “Fast algorithms for geometric traveling salesman problems.”, ORSA Journal on Computing, 1992, issn: 08991499. [Online]. Available: http: / / search . ebscohost . com . focus . lib . kth . se / login . aspx ? direct = true db = bsh AN = 4472271site=ehost-live.

[Online]. Available: http : / / en . wikipedia . org / wiki / Travelling _ salesman _ problem.

[Online]. Available: https://tsp-basics.blogspot.com/