**Optimizing real road networks by applying time efficient shortest path algorithms**

A Synopsis Submitted

in Partial Fulfillment of the Requirements

for the Course of

# Minor Project - I

In

Third year – Fifth Semester of

**Bachelor of Technology**

specialization

In

# Business Analytics and Optimization

Under

**Mr.Amit Singh**

By

**500067103 R103218043 Divyam Verma**

**500069069 R103218081 Mayank Rastogi**

**500067983 R103218037 Devansh Messon**



DEPARTMENT OF INFORMATICS

SCHOOL OF COMPUTER SCIENCE

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES, BIDHOLI, DEHRADUN, UTTRAKHAND, INDIA

# Jan, 2020

Synopsis

1. **Introduction**

Compute the shortest route possible from a particular source location to one or more destinations in least possible time by optimizing and applying time efficient shortest path algorithms.

If there is only one destination to visit then the aim is to determine the shortest possible path from source to destination.

If there are multiple destinations to visit then the aim to determine a route such that the vehicle visits all the destinations and comes back to the source location in the least possible time. Objective is to order the destinations such that vehicle visits the destinations in that order.

In this pandemic, many people encountered various lethal health problems, therefore it becomes an emergency case and such situation expects an ambulance to reach the patient’s place as soon as possible. Obviously, ambulance must take the shortest possible path from the hospital to patient’s place in the real road network given in the form of a graph to reach the destination faster to save a life.

Our objective is to suggest the ambulance a shortest possible path as soon as possible in the real road network by applying time efficient shortest path algorithms.

If there are more than one emergency cases at a time,then ambulance is required to visit all places(patient’ home) in the fastest possible way.So Travelling Salesman Problem(TSP) can be applied to provide the shortest possible route that visits each place and returns back to hospital.

If there is only one emergency case at a time, then ambulance has only one destination to reach.In this case, dijkstra algorithm can be applied to provide the shortest path from the hospital to the patient’s home.

TSP and dijkstra algorithm take much time (execution time of algorithm) to compute the shortest possible path when the graph is complex and large. In that case, heuristic/nature-inspired optimization technique can be a good solution to optimize(In terms of time complexity) the TSP and dijkstra solution which results in decreasing the execution time and shortest possible path is computed relatively faster despite of a complex and large graph.

1. **Motivation**

Travelling Salesman Problem is implemented using:

-Back-Tracking Method:

Time-Complexity-O(v!) , where v is the number of vertices..

-Dynamic Programming Method:

Time Complexity- O(v2 \* 2v), where v is the number of vertices.

As value of n may be large. So,Time Complexity of above mentioned methods

are very high.To overcome this problem, TSP is needed to be optimized further.

Dijkstra Algorithm is implemented using:

- Bruteforce method:

Time-Complexity-O(v2) , where v is the number of vertices.

-Binary heap implementation:

Time-Complexity-O(ELogV), where v is the number of vertices and E is the number of edges.

Time-Complexity of above mentioned methods needs to be reduced further.

1. **Related work**

**Travelling Salesman Problem** is Optimized by a nature inspired technique named Ant Colony Method.

The main idea is to partition artificial ants into two groups: scout ants and common ants. The common ants work according to the search manner of basic ant colony algorithm, but scout ants have some differences from common ants, they calculate each route’s mutation probability of the current optimal solution using path evaluation model and search around the optimal solution according to the mutation probability. Simulation on TSP shows that the improved algorithm has high efficiency and robustness.

**Dijkstra Algorithm** is further Optimized by an algorithm named Bi-Directional Dijkstra Algorithm.

In Bi-directional Dijkstra Algorithm, Dijkstra Algorithm runs from the both source node and destination node simultaneously. If a shortest path exists between source and destination node, then both searches will meet at an intermediate vertex.

Bi-directional Algorithm runs 2 times faster than the Dijkstra Algorithm.

1. **Proposed Method**

Implementing the Related work mentioned above and comparing the time complexities of Standard TSP and Dijkstra algorithm with the optimized algorithm.

1. **Methodology**

**-TSP**

**Backtracking Method:**

1.Root node or source node(1) is the starting point and ending point of the final path.

2.Generate all (n-1)! permutations of the nodes as:

-Traverse the array of nodes by variable i.

-Swap starting node and ith node.

-Recursively call the function with starting node=starting node+1.

-Swap starting node and ith node.

3.when starting node==ending node,Calculate cost of the path.

4.Return the permutation having the minimum cost.

**Dynamic Programming Method:**

1. Select a vertex from a set of vertices( of an undirected graph)
2. Consider the selected vertex as a root node
3. For every other vertex i(except root node ) find the minimum cost path (cost(i)) with root node as the starting point and I as the ending point
4. The minimum cost path(cost(i)) should contains all the vertices for exactly once
5. The cost of corresponding cycle would be cost(i) + dist(i,1) [where dist(i,1) is the distance from i to 1]
6. Find/return the minimum of all [cost(i)+dist(i,1)]values and calculate cost(i)using Dynamic Programing
7. Have recursion relation in terms of sub problems of the graph

**Dijkstra Algorithm:**

**-Bruteforce method**

(a)Extracting the vertex with minimum distance by running a O(n) loop.

(b)Relaxing the vertices adjacent to the picked vertex in O(1).

(c)Repeat it (a) and (b) n times.

**-Binary heap implementation**

(a)Building a minimum heap containing all the unvisited vertices.

(b)Extracting the minimum vertex from root of heap.

(c)Deleting the picked vertex from heap in O(Log N)

(d)Relaxing the adjacent vertices of picked vertex in O(Log N).

(c).Repeat the above steps till heap gets empty.

**bidirectional implementation**

1.Build forward binary heap [source=original source of the input graph]

2.Build backward binary heap [source=destination of input graph.]

3.while(1)

{

4. Run Dijkstra algorithm from source(forward dijkstra)

5.find the linking point i=top node

6. final\_distance=min(final\_distance , forwardist[i] + backwardist[i])

7.if(visit[top node]==1)break; //Termination Condition

8.else visit[i]=1;

9.Run Dijkstra algorithm from destination(backward dijkstra)

10.find the linking point i=top node

11.final\_distance=min(final\_distance , forwardist[i] + backwardist[i])

12.if(visit[top node]==1)break; //Termination Condition

13.else visit[i]=1;

14.if(forward heap is empty || backward heap is empty)break;

}

**Fibonacci heap implementation**

**1.Building fibonacci heap**

(a).Allocate a new node.

(b).Make its parent,child,left,right as NULL.

(c.)degree,mark=0

(d)Distance of nodes from source node is infinity except source node.

(e)If heapsize==0:

(f)Minimum element=input

(g)if heapsize>0:

(h)Minimum element=min(Minimum element,input)

(i)Time-Complexity-O(n)

**2.Deleting Minimum element from Fibonacci heap**

(a)Unlink the minimum node from the root list and

(b)add all its children to the root list.

(c)Degree of node=number of children.

(d)All Subtrees should have unique degree.

(e)Max. Degree in the Root list= Log2 (N)

(f)Declare array pointer which is pointing to node which has degree=array’s index

(g)Traverse the root list of Fibonacci heap.

(h)If array[current node’s degree]==NULL:

(i)then array[current node’s degree] will point to current node.

(j)If array[current node’s degree]!=NULL:

(k) If array[current node’s degree] value > current node value

(l) then make current node as parent of array[current node’s degree]

(m) array[current node’s degree] will point to current node

(n)else vice versa

Worst case Time Complexity-O(nLogn)

**3.Implementation of Deleting Minimum element from Fibonacci heap continued**

**4.After every subtree has unique degree in the Fibonacci heap.**

**5.Traverse the array pointer and find the minimum element of the Fibonacci heap**

**Plan of Work :**

**1. Genetics algorithm to optimize TSP**

**References:**

**[1]** [**https://www.youtube.com/**](https://www.youtube.com/)

**[2]https://www.geeksforgeeks.org**

**[3] CLRS Book**