Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering

by

Shivansh Mani Tripathi

MIS No:112015131

Semester: IV

Computer Science and Engineering

Indian Institute of Information And Technology Pune

Near Bopdev Ghat, Kondhwa Annexe, Yewalewadi, Pune, Maharashtra 411048

April 2022

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem"** submitted by **Shivansh Mani Tripathi** bearing the **MIS No:112015131**, in completion of his project work under the guidance of **DR Ranjith Nair** is accepted for the project report submission in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering , Indian Institute of Information Technology, Pune, during the academic year 2020-21.

Dr Ranjith nair
Project Guide
Assistant Professor
Department of CSE
IIIT Pune

Dr. Tonmoy Hazra
Head of the Department,CSE
Assistant Professor
Department of CSE
IIIT Pune

Project viva-voce held on 30/04/2022

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

This project would not have been possible without the help and cooperation of many. I would like to thank the people who helped me directly and indirectly in the completion of this project work.

First and foremost, I would like to express my gratitude to our beloved director, **Dr. Anupam Shukla**, for providing his kind support in various aspects.

I would like to express my gratitude to my project guides **Dr. Ranjith Nair**, Assistant Professor, Department of CSE, for providing excellent guidance, encouragement, inspiration, constant and timely support throughout this B.Tech project.

I would like to express my gratitude to the head of department **Dr. Tonmoy Hazra**, Head of Department CSE, for providing his kind support in various aspects.

I would also like to thank all the faculty members in the Dept. of CSE and my classmates for their steadfast and strong support and engagement with this project.

Abstract

Finding the shortest path in direction effective is essential. To solve this shortest path problem, we usually using Dijkstra or A* algorithm. These two algorithms are often used in routing or road networks. This paper's objective is to compare those two algorithms in solving this shortest path problem. In this research, Dijkstra and A* almost have the same performance when using it to solve town or regional scale maps, but A* is better when using it to solve a large scale map.

TABLE OF CONTENTS

Chapter 1

Introduction

Technology has been helping us a lot, from sending email, text messaging, social media, and even maps. People using maps for many things, e.g., searching for places such as searching for the nearest restaurant, café, gas station, or finding direction from home to work. When requesting a route from one point (starting point) to another location (destination point), usually, the result that comes out is the "shortest path" from starting point to destination point. The shortest path problem has been studied for many years. The shortest path problem is the problem that finds the minimum distance or pathway between nodes or vertices in a graph (for this case, road network). A graph is an abstract mathematical object, which contains sets of vertices and edges .

Regarding to journal from Parveen Sharma and Neha Khurana, the shortest path problem is defined as that of finding a minimum-length (cost) path between a given pair of nodes and according to Karishma Talan and G. R. Bamnote, shortest path problem is the problem of looking for the quickest way to get route from one location to another .

Expressed more formally, in a graph in which vertices are joined by edges and in which each edge has a value or cost, it is a problem of finding the lowestcost path between two vertices .

Distribution is one of marketing activities in aim to facilitate the producers in sending goods to the customers. In the distribution process, a set of customers is served by a producer where located in an area. A set of crews operates a set of vehicles which is located in the producer place called a depot to fulfill the each customer demand. As a producer, for minimizing their total operational transportation cost of distribution, the

depot has to find an appropriate road network for the vehicles .

Many algorithms solve the shortest path problem. Dijkstra's algorithm is one form of the greedy algorithm. This algorithm includes a graph search algorithm used to solve the shortest path problem with a single source on a graph that does not have a negative side cost and produces the shortest path tree. This algorithm is often used in routing .

A heuristic is a method designed for solving a problem more quickly. This is achieved by trading accuracy, optimality, completeness, or precision for speed. A* algorithm is an algorithm that finds a path in a graph from a given initial node to a given goal node. It employs a "heuristic estimate" h(x) that gives an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate .

This paper will compare Dijkstra's algorithm and A* algorithm to solve the shortest path problem. There are four stages in mathematical modelling: problem identification, constructing a mathematical model, determining mathematical solution from the model, and interpreting mathematical solution into real problem points of view .

Chapter 2

Motivation

2.0.1 OBJECTIVE

This paper's objective is to compare those two algorithms in solving this shortest path problem. In this research, Dijkstra and A* almost have the same performance when using it to solve town or regional scale maps, but A* is better when using it to solve a large scale map.

2.0.2 MOTIVATION

Due to the different purposes the travel of the passengers and its transport requirements are different. For example, for business travel, the passenger wants the time of travel as short as possible. Tourists prefer the cost of travel as minimum as possible. National City traffic advisory procedures can satisfy the passengers demand and provide two to three optimal decision rules according to the transport advisory for passengers. The procedures allow users to look up city information, choose the optimal decision rule and edit city information. At first, the users choose the starting station, the terminal station, the optimal decision rules and traffic tool. The output information are the most time-efficient or the cost- efficient decision, the cost of time and money. Users can also add the city information and edit the cost and the timetable of traffic tools.

Chapter 3

Literature Review

3.0.1 Literature Review

Regarding to journal from Parveen Sharma and Neha Khurana, the shortest path problem is defined as that of finding a minimum-length (cost) path between a given pair of nodes and according to Karishma Talan and G. R. Bamnote, shortest path problem is the problem of looking for the quickest way to get route from one location to another . Expressed more formally, in a graph in which vertices are joined by edges and in which each edge has a value or cost, it is a problem of finding the lowestcost path between two vertices .

3.0.2 Refrences

1- Kairanbay M, and Mat Jani H 2013 A Review And Evaluations Of Shortest Path Algorithms. Int. J. of Sci. Tech.Res.2 6 99.

2- Sharma P, and Khurana N 2013 Study of Optimal Path Finding Techniques. Int. J. of Adv.in Tech. 4 2, 124.

3- Talan K, and Bamnote G R 2015 Shortest Path Finding Using a Star Algorithm and Minimum Weight Node First Principle. Int. J. of Innovative Res. in Computer and Communication Engineering 3 2 1258.

4- Hart E, Nilsson N J and Raphael, B 1968 A Formal Basis for the Heuristic Determination of Minimum Cost Paths Systems Science and Cybernetics 4 2 100-107.

.

Chapter 4

Methodology with Dataset Analysis

4.0.1 Methodology

In this implementation, we will use 2 examples from the University of Sumatera Utara at Jl. Universitas No. 9-A (starting point) to Wisma Benteng Restaurant at Jl. Kapt. Maulana Lubis No. 6 (1st destination point) and from the University of Sumatera Utara at Jl. Universitas No. 9-A (starting point) to the State University of Medan at Jl. William Iskandar Ps. V (2nd destination point). Between the starting point and destination point, we manually generate plenty of points (nodes), which is an intersection between 2 or more roads and find the distance between them (edges). After that, we apply the data to Dijkstra's Algorithm, A* algorithm, and make a comparison between the two algorithms based on the running time, the number of loops with the number of points as the factor. The processes are as below. 2.1 Steps for getting the distance between intersection and distance between point to destination point: First, we need to manually generate point which is the intersection between 2 or more roads that is possible to be crossed between starting point to both destination points (1st destination point is marked with green color and 2nd destination point is marked with orange color). We use Google Maps or Geographic Information System to map the point, as shown below.

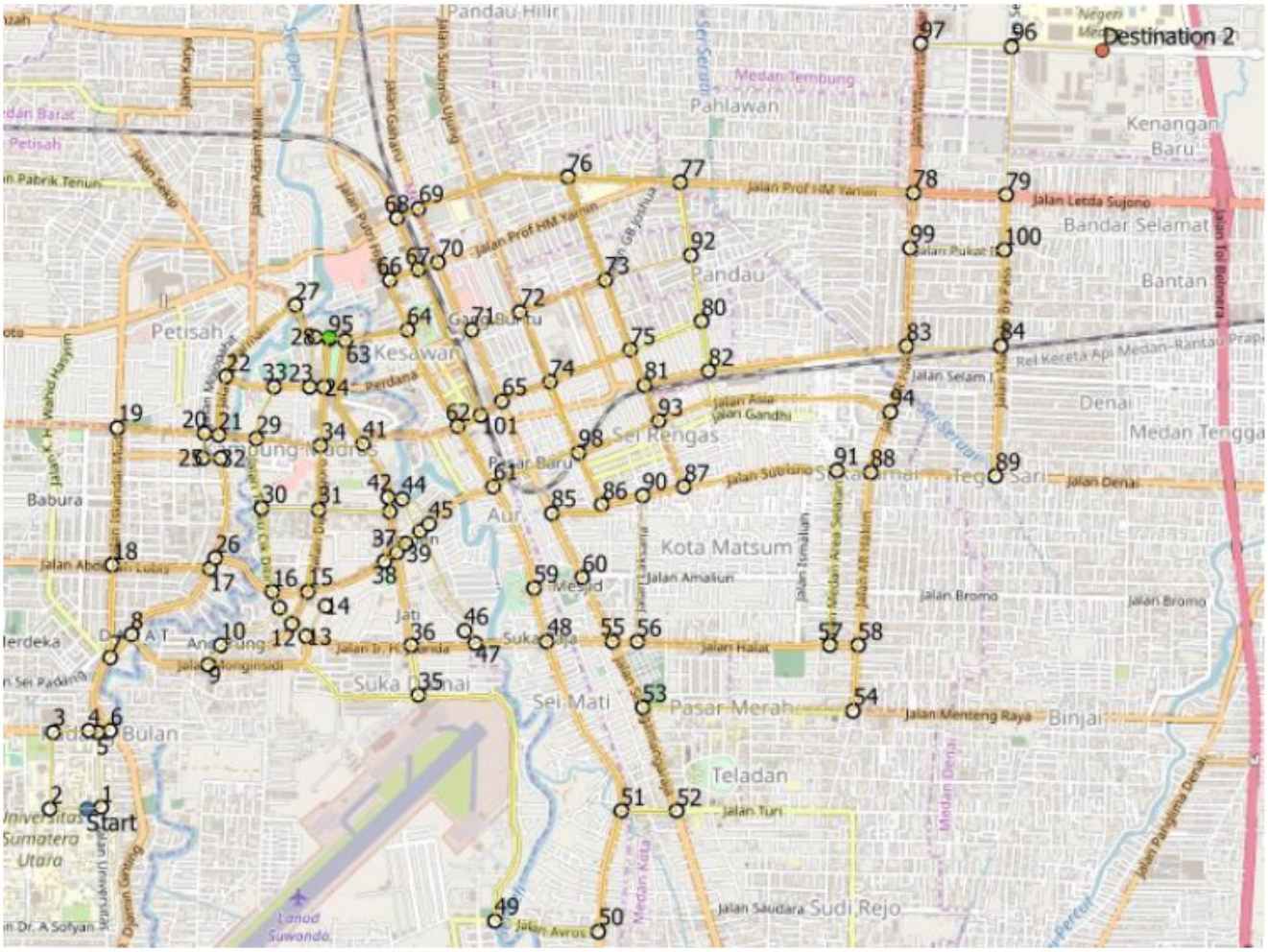

Figure 1. Mapping all the possible intersection with map background

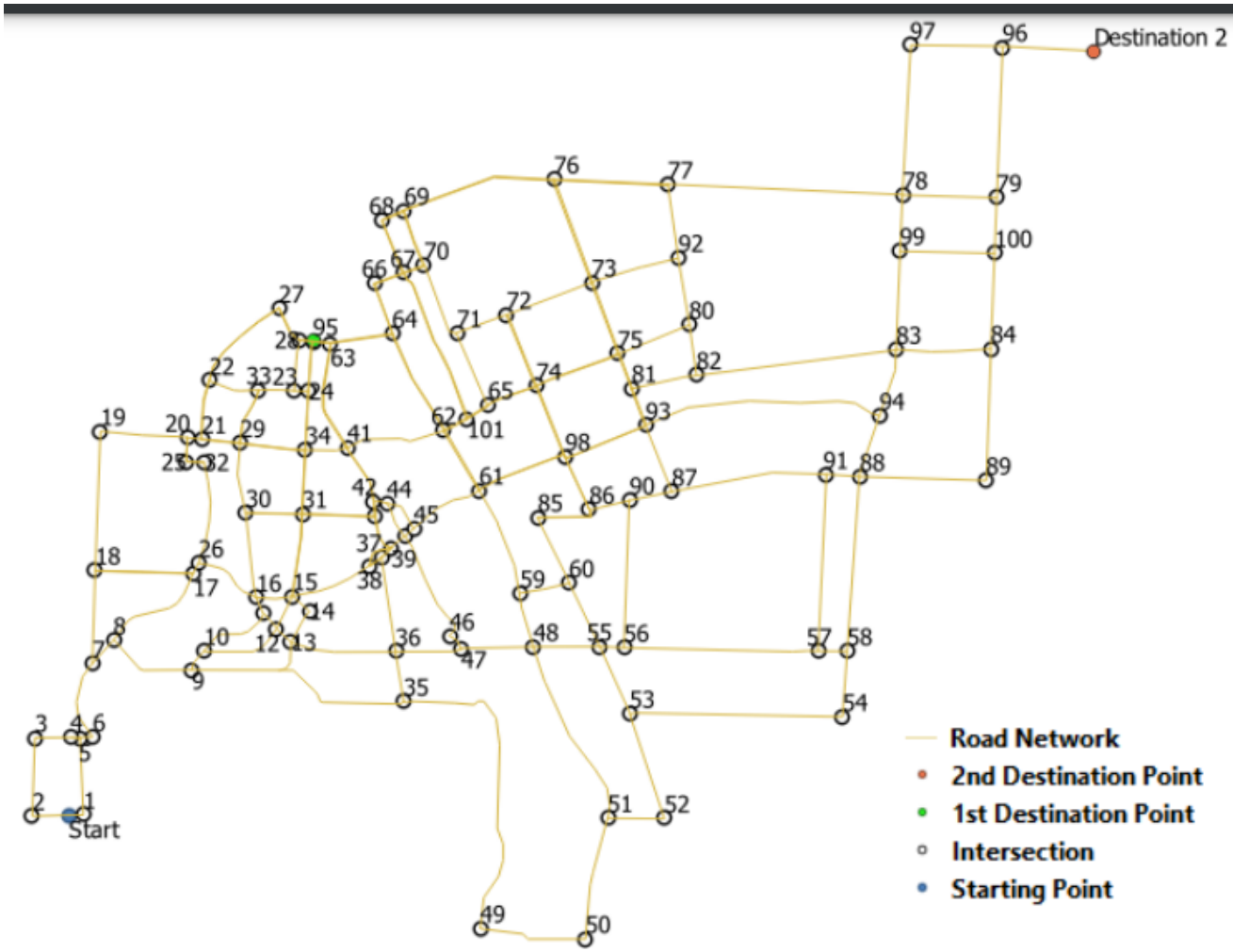

Figure 2. Mapping all the possible intersection without map background

After that, we calculate the distance between points connected by the road and estimate the distance between point to the destination point for heuristic value by using the help from Google Maps.

Table 1. Distance between points connected by the road

From point	To point	Distance / Weight (in km)
start	2	0.23
start	1	0.084
1	5	0.5
2	3	0.5
…	…	…
101	65	0.14

Table 2. Distance between point to 1st destination point

From	Distance / Weight (in km)
1	3.283108
2	3.441482
3	3.018227
4	2.889003
…	…
34	0.677927

Table 3. Distance between point to 2nd destination point

From	Distance / Weight (in km)
1	7.91655
2	8.185879
3	7.889952
4	7.696507
…	…
101	4.543907

2.2 Steps for calculating the shortest path:

Dijkstra's Algorithm steps:

1. Set all points distance to infinity except for the starting point set distance to 0.

2. Set all points, including starting point as a non-visited node.

3. Set the non-visited node with the smallest current distance as the current node "C."

4. For each neighbor "N" of your current node: add the current distance of "C" with the weight of the edge connecting "C" – "N." If it's smaller than the current distance of "N," set it as the new

current distance of "N." 5. Mark the current node "C" as visited.

6. Repeat the step above from step 3 until the destination point is visited. A* algorithm is just like Dijkstra's algorithm, and the only difference is that A* tries to look for a better path by using a heuristic function, which gives priority to nodes that are supposed to be better than others while Dijkstra's just explore all possible ways.

A* algorithm steps:

1. Set all point distance to infinity except for the starting point set distance to 0.

2. Set all points, including start point as a non-visited node. 3. Set the non-visited node with the smallest current distance as the current node "C."

4. For each neighbor "N" of your current node: add the current distance of "C" with the weight of the edge connecting "C" – "N" and the weight to the destination point (heuristic). If it's smaller than the current distance of "N," set it as the new current distance of "N."

5. Mark the current node "C" as visited.

6. Repeat the step above from step 3 until one of the neighbors "N" is the destination point.

We build the Dijkstra's Algorithm program and A* program as shown below. After the program finish, we input all the data from the previous step to the program.

```
while queue:
    queue_count = queue_count + 1 #loop count
    # find min distance which wasn't marked as current
    key_min = queue[0]
    min_val = path[key_min]
    for n in range(1, len(queue)):
        if path[queue[n]] < min_val:
            key_min = queue[n]
            min_val = path[key_min]
    cur = key_min
    queue.remove(cur)

    for i in graph[cur]:
        alternate = graph[cur][i] + path[cur]
        if path[i] > alternate:
            path[i] = alternate
            adj_node[i] = cur
```

Figure 3. Dijkstra's Algorithm Source Code

```
while queue and status == 1:
    queue_count = queue_count + 1 #loop count
    # find min distance which wasn't marked as current
    key_min = queue[0]
    min_val = path_heu[key_min]
    for n in range(1, len(queue)):
        if path_heu[queue[n]] < min_val:
            key_min = queue[n]
            min_val = path_heu[key_min]
    cur = key_min
    queue.remove(cur)
    for i in graph[cur]:
        alternate = graph[cur][i] + path[cur]
        alternate_heu = graph[cur][i] + path[cur] + heu[i]
        if path_heu[i] > alternate_heu:
            path_heu[i] = alternate_heu
            path[i] = alternate
            adj_node[i] = cur

            if 'FINISH' in graph[cur]:
                #while_loop_break
```

Figure 4. A* Algorithm Source Code

Chapter 5

Results and Discussion with dataset description

5.0.1 Results and Discussion

The experiments were performed on laptop Windows 10 pro with 8GB of RAM and Intel i5-6200 processor with PyCharm version 2018.3.5 64-bit version evaluation copy. 3.1. Starting point to 1st destination point (Total of 36 points include start and destination point) Running Time We take 3 tries for each Dijkstra and A*, running times are shown below

Table 4. Running Time from starting point to 1 destination point using Dijkstra and A* algorithm

Test Number	Running Time Dijkstra (nanoseconds)	Running Time A* (nanoseconds)	Total Distance (in km)
1	997.200	996.800	4.504
2	998.200	996.900	4.504
3	997.800	997.400	4.504

Loop Counts
We also take 3 run of each algorithm and the result shown below.

Table 5. Loop counts from starting point to 1st destination point using Dijkstra and A* algorithm

Test Number	Loop Count Dijkstra	Loop Count A*	Total Distance (in km)
1	36	26	4.504
2	36	26	4.504
3	36	26	4.504

3.2. *Starting point to 2nd destination point* (Total of 103 points include start and destination point)
Running Time
We also take 3 tries for each Dijkstra and A*, running times are shown below.

Table 6. Running Time from starting point to 2nd destination point using Dijkstra and A* algorithm

Test Number	Running Time Dijkstra (nanoseconds)	Running time A* (nanoseconds)	Total Distance (in km)
1	2,991.400	2,992.000	10.138
2	3,987.800	2,992.800	10.138
3	3,989.100	2,992.100	10.138

Loop Counts
We also take 3 run of each algorithm and the result shown below.

Table 7. Loop counts from starting point to 2nd destination point using Dijkstra and A* algorithm

Test Number	Loop Count Dijkstra	Loop Count A*	Total Distance (in km)
1	103	72	10.138
2	103	72	10.138
3	103	72	10.138

Chapter 6

Conclusion and Future Work

6.0.1 Conclusion and Future Work

In conclusion, the use of Dijkstra's algorithm and A* algorithm in the shortest path is essential will give the same output in no time when being used on the town or regional scale maps. But on a large scale map, A* will provide the solution faster than Dijkstra. A* scan the area only in the direction of destination because of the heuristic value that counted in the calculation, whereas Dijkstra searches by expanding out equally in every direction and usually ends up exploring a much larger area before the target is found resulting making it slower than A*. This can be proven by the loop count of Dijkstra and A*, the more points (nodes) the higher the difference between the loop count nor the time.