



A realistic shortest path considering road intersections.

Md. Abu Naeem

171-35-1868

A Thesis submitted in partial fulfillment of the requirement for
the degree of Bachelor of Science in Software Engineering

Department of Software Engineering,

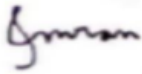
DAFFODIL INTERNATIONAL UNIVERSITY

Fall'2020

APPROVAL

This Thesis titled “A realistic shortest path considering road intersections.”, submitted by **Md. Abu Naeem (171 – 35 - 1868)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of BSc. in Software Engineering and approved as to its style and contents.

BOARD OF EXAMINERS



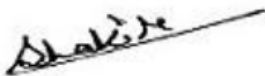
Dr. Imran Mahmud
Associate Professor and Head
Department of Software Engineering
Daffodil International University

Chairman



Md Anwar Hossen
Assistant Professor
Department of Software Engineering
Daffodil International University

Internal Examiner 1



Asif Khan Shakir
Senior Lecturer
Department of Software Engineering
Daffodil International University

Internal Examiner 2



Professor Dr M Shamim Kaiser
Institute of Information Technology
Jahangirnagar University

External Examiner

DECLARATION

I hereby, declare that I have taken this thesis under the supervision of Mr. K.M. Imtiaz-Ud-Din, Assistant Professor, Department of Software Engineering, Daffodil International University. I also admit that neither this thesis nor any part of this has been submitted elsewhere for award of any degree.

Submitted By



.....

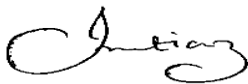
Md. Abu Naeem

ID: 171 – 35 - 1868

Batch: 22th

Department of Software Engineering, Faculty of Science and Information Technology,
Daffodil International University.

Certified by:



.....

K.M. Imtiaz-Ud-Din

Assistant Professor, Department of Software Engineering,

Faculty of Science and Information Technology,

Daffodil International University.

ACKNOWLEDGEMENT

Throughout Writing this of thesis I faced difficulties and problems to solve. But with the help of our respectable supervisor sir, I overcame all difficulties and solved every problem I faced through this journey. Through this journey I learned a lot

First of all, I am grateful to my almighty Allah for making me capable enough to complete this thesis.

And I really want to thank my respectable supervisor sir, Mr. K. M. Imtiaz -Ud-Din, Assistant professor, Department of Software Engineering, Daffodil University for his continuous help through this journey.

Also, thanks to my department for enhancing our skill though this Final Project/thesis. Hopefully this will help us in our upcoming future.

Abstract

Finding shortest path plays an important role in many areas. Such as Robotics, Road maps, Network communications etc. There are some popular algorithms that can find the shortest path. In different environments, our conventional algorithm needs to be modified to solve a specific problem. In this paper we modified the conventional Dijkstra algorithm to find the shortest path considering intersections. We tested the performance of our improved algorithm on a real map which is collected from Google map. The algorithm is implemented using Visual C++14, and our analysis results showed that this algorithm is effective and more accurate in real life.

Key Words – Shortest path, Improved Dijkstra algorithm, Intersections.

Table of Contents

| | |
|---|-----|
| ACKNOWLEDGEMENT | iii |
| Abstract | iv |
| LIST OF FIGURES | vi |
| LIST OF TABLES | vi |
| CHAPTER 1 | 1 |
| INTRODUCTION | 1 |
| 1.1 Background | 1 |
| 1.2 Motivation | 1 |
| 1.3 Problem Statement | 1 |
| 1.4 Research Question | 2 |
| 1.5 Research Scope | 2 |
| CHAPTER 2 | 3 |
| LITERATURE REVIEW | 3 |
| 2.1 Existing Works | 3 |
| CHAPTER 3 | 4 |
| PROPOSED METHODOLOGY | 4 |
| 3.1 Relevant Algorithm | 4 |
| 3.2 justifying algorithm | 4 |
| 3.2 Conventional Dijkstra Algorithm | 5 |
| 3.4 Improved Dijkstra Algorithm | 6 |
| CHAPTER 4 | 10 |
| RESULTS AND DISCUSSION | 10 |
| 4.1 Analysis | 10 |
| Chapter 5 | 16 |
| CONCLUSION | 16 |
| REFERENCES | 17 |
| APPENDIX A | 18 |
| Pseudo code: Improved Dijkstra Algorithm | 18 |
| APPENDIX B | 19 |
| Accounts Clearance | 19 |

LIST OF FIGURES

| | |
|--|----|
| Figure 3.3.1 Graph 1 ----- | 11 |
| Figure 3.4.1 Graph 2 ----- | 13 |
| Figure 4.1.1 A real map taken from Google Map which represents Dhaka University area and surroundings, Dhaka, Bangladesh ----- | 15 |
| Figure 4.1.2 Output 1 ----- | 16 |
| Figure 4.1.3 Output 2 ----- | 17 |
| Figure 4.4.1 Output 3 ----- | 17 |

LIST OF TABLES

| | |
|--|----|
| Table 3.4.1 The Path Search Process of Our Improved Dijkstra Algorithm ----- | 14 |
|--|----|

CHAPTER 1

INTRODUCTION

1.1 Background

Path planning is a very common problem in computer science. It has many useful application areas like artificial system, network, automatic guided vehicle etc. In modern computer science path planning plays an important role. This research helps to find the best path from one point to another.

In decades many path planning algorithms has been proposed such as Dijkstra algorithm [1], A* algorithm [2]. Etc.

1.2 Motivation

During my Literature review, I find many of the authors present some interesting problems and solutions on behalf of this shortest path algorithm. There are still some deficiencies in this area to talk about. It is a great opportunity to contribute my knowledge in this field.

Relevant present article makes the following contributions

- Path planning for **Robots**.
- Path planning considering **Traffic Condition**.
- Path planning considering **Lane Changing**.
- Path planning considering **U-Turn**.

1.3 Problem Statement

If we want to travel from a source node to a destination node and there are many paths we can use. In this situation the best path would be a path which takes less time to travel among all of them.

Generally, the length of the path is the main reason to determine the travel time. We can choose a path among many which has less path length. but if we consider the real world, we can understand the length of the path is not the only reason to determine the travel time of a path.

This paper analyzes some graphs with nodes and edges including weight. And we consider graphs intersections in major. Because intersections can play an important role to increase the travel time of a path. For a road map, we know in intersections the traffic condition is worse than single roads. In intersections two or more roads are connected in a single node/intersection that increases the pressure of the vehicle in a single node. So that a vehicle takes more time to pass that intersection.

What if we find a path which has less intersections among all other existing paths? In this paper we are trying to figure out the solutions for that.

1.4 Research Question

Q₁ - How it works when we consider Path intersections.

Q₂ - Can we reduce travel time more accurately Considering Path intersections?

1.5 Research Scope

As our research is on the shortest path algorithm, this algorithm has many applications in the real world.

Road network is one of the most common area where we can use this algorithm. If we need to find a way from one point to another point, then we can use this algorithm. Using this algorithm, we can find the best path among all other paths. Also, we can design new road networks for a city.

Routing plays a very important role in network communications. If we want to send data from a computer to another through an online network, we have to use multiple routers to carry our data. Since there are so many routers in the network so there are a lot of existing valid paths from a point to another point. So, we have to find the best path among all of them. Path finding algorithms can solve this problem.

Robotic path planning can make robots smart and efficient. Using this algorithm, robots can find the best path for travelling. Because of this algorithm, Robot can skip its obstacle.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing Works

As our research topic is “A realistic shortest path considering road intersections” we tried to review some latest Relevant Papers. We got some interesting papers where authors proposed their problem and gave appropriate solutions.

We know automated guided vehicles technology is fast growing technology. In a rectangular environment map our conventional algorithm can find out only one path and skip others paths with equivalent length. To solve this problem an improved Dijkstra algorithm can find out all equivalent shortest paths [3].

Traffic conditions in a common scenario all over the world. Usually In city traffic, traffic problems are a big problem because of the huge number of vehicles. For large amounts of requests, it needs to be solved as fast as possible. So, using the Dijkstra algorithm this problem can be solved with the shortest path and also alternative paths according to traffic conditions [4].

An improved Dijkstra Algorithm was proposed to solve 2D Grid Maps. The algorithm tried to ensure that each position needs to calculate once to get the shortest distance to destination [5].

There is a new strategy which is called Aggressive heuristic search. It finds a solution instantly, after that if time permits it can find a bounded suboptimal solution [6].

A different environment was created for solving the robot's problem. A modified parameter in the Dijkstra algorithm was implemented in a robot. And tried to find a best path reaching the destination from the initial position of the robot [7].

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Relevant Algorithm

During the decades, many algorithms have been proposed as variants of the shortest path algorithm. Like, Breadth First Search (BFS), Dijkstra Algorithm, Floyd-Warshall Algorithm, bellman Ford Algorithm etc.

- ✓ **BFS** is a traversing algorithm. It starts with the root node. Then explore every other node level wise. It works on Unweighted Graph. Time complexity is $O(V + E)$.
- ✓ **Dijkstra algorithm** is a single source shortest path finding algorithm. It works on a weighted Graph. Time complexity is $O(V \log V + E)$.
- ✓ **Bellman Ford algorithm** calculates the shortest paths from a single source node to all of the other nodes in a weighted digraph. Time complexity is $O(V * E)$, It's slower than the Dijkstra algorithm for the same problem. But it can handle Negative Weighted Graphs.
- ✓ **Floyd-Warshall** uses it for finding the shortest paths in a directed weighted graph with positive/ negative weights but with no negative cycles. Time complexity is $O(V^3)$.

Here, V is a set of nodes, E is a set of Edges.

3.2 justifying algorithm

In this paper, I am working on the Dijkstra Algorithm. Our conventional Dijkstra Algorithm can find the shortest path among existing paths. And for finding the best, it considers the weight of the path. Best path is declared when the chosen path has less total weight. Since I want to consider intersections besides the weight of edges. So, I am going to improve our conventional Dijkstra Algorithm so that it can solve my proposed problem.

3.2 Conventional Dijkstra Algorithm

Dijkstra algorithm is a more classical algorithm than all the shortest path algorithms.

Suppose you are traveling and want to know which way to go from U to V will be the fastest. There are now several roads from node U. The shortest path is only one road. So, the Dijkstra algorithm can help you to find this shortest path on the computer.

Let's go to work now. The Dijkstra Algorithm is called Single Source Shortest Path Algorithm. That's why, with this algorithm we can find out the minimum cost to go from a specific node to all nodes.

The Dijkstra algorithm is a greedy algorithm. Considering a directed graph $G = [V, E]$ with n nodes and m edges. Where V is set of nodes, E is set of Edges. $C[u][v]$ denotes the weight of edge $(u - v)$. if edge $(u - v)$ does not exist, Then $C[u][v] = \text{infinite}$. $\text{dist}[x]$ conveys the distance from the source node s to the node x .

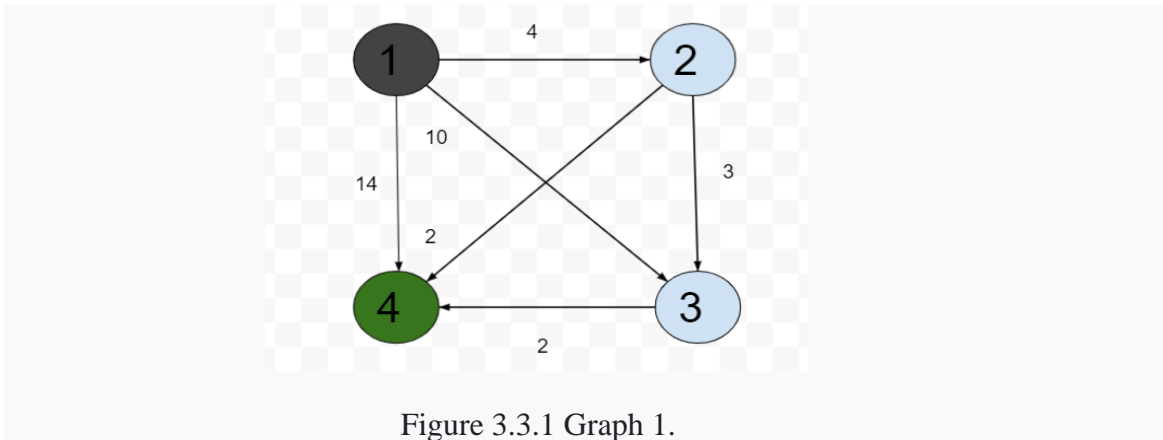
S represents the set of nodes that is included in the shortest path.

Process is as follows:

1. Initialization. We have pushed our source node to the queue. $S = \{s\}$;
2. We took a loop until the queue was empty and each time, we pulled out the node (u) in front of everyone in the queue.
3. Then, check in another loop if one of its adjacent nodes (v) has a weight of the connecting edge $C[u][v]$ is exists.
4. If $\text{dist}[v] > \text{dist}[u] + C[u][v]$, the distance passing through the node u is shorter than $\text{dist}[v]$, modify $\text{dist}[v]$ to $\text{dist}[v] = \text{dist}[u] + C[u][v]$, and add node v to S .
5. Repeat step (3) and (4) $n - 1$ time.

The shortest path from the source node to goal node is calculated.

For Example, find a shortest path source node (1) to goal node (4) in a directed graph shown in Figure 3.3.1



Our resulted shortest path using Dijkstra algorithm is $1 > 2 > 4$.

3.4 Improved Dijkstra Algorithm

Considering a directed graph $G = [V, E]$ with n nodes and m edges. Where V is set of nodes, E is set of Edges. $C[u][v]$ denotes the weight of edge $(u - v)$. if edge $(u - v)$ does not exist, Then $C[u][v] = \text{infinite}$. $\text{dist}[x]$ conveys the distance from the source node s to the node x . $\text{Degree}[x]$ represents the number of connected roads in the intersection identified by x . $\text{total_Deg}[x]$ represents the number of connected roads in all intersections through the path of source node to node x .

S represents the set of nodes that is included in a shortest path.

Process is as follows:

1. Initialization. We have pushed our source node to the queue. $S = \{s\}$;
2. We took a loop until the queue was empty and each time, we pulled out the node (u) in front of everyone in the queue
3. Then, check in another loop if one of its adjacent nodes (v) has a weight of the connecting edge $C[u][v]$ is exists.
4. If $(\text{dist}[v] > \text{dist}[u] + C[u][v] + \text{Degree}[u])$ OR $(\text{dist}[v] == \text{dist}[u] + C[u][v] + \text{Degree}[u]$ AND $\text{total_Deg}[u] + \text{Degree}[u] < \text{total_Deg}[v])$,

The distance passing through the node u is shorter than $\text{dist}[v]$ Or if the distance of two different paths is equals. For second conditions, takes the path which have less connected roads in intersections through the path.

If one of these conditions is true,

- ✓ Modify $\text{dist}[v]$ to $\text{dist}[v] = \text{dist}[u] + C[u][v] + \text{Degree}[u]$;
- ✓ Modify $\text{total_Deg}[v] = \text{total_Deg}[u] + \text{Degree}[u]$;
- ✓ Add node v to S .

5. Repeat step (3) and (4) $n - 1$ time.

The shortest path from the source node to goal node is calculated.

In the graph shown in Figure 3.4.1, Consider an undirected graph $G = [V, E]$ with 8 nodes and 9 edges. V is a set of nodes and E is a set of Edges. “Used” is the set of the nodes included in the shortest paths. “Unused” is the set of other nodes which are not included in any shortest path. $d(x)$ denotes the total number of connected roads passes through all intersections from the root node to node x . $w(x)$ denotes the total calculated weights from root node to node x . Our algorithm, finding out the shortest path from root node 1 to all nodes considering intersections and weight of edges.

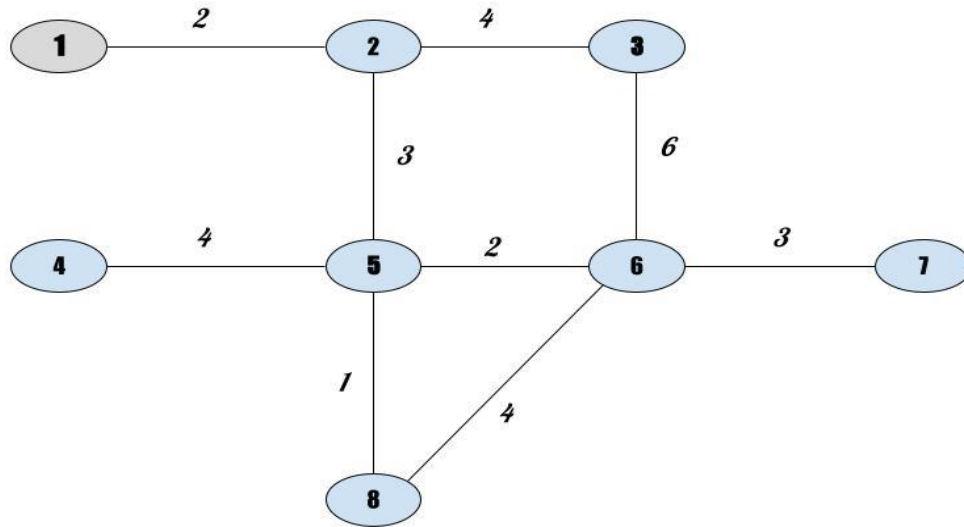


Figure 3.4.1 Graph 2.

The search process is shown in Table 3.4.1.

| Step | Used Set | Unused Set |
|------|---|--|
| 1 | Initial Used = {1} Shortest Path: $1 - 1 = d(1) + W(1) = 0 + 0 = 0$. Search from 1 | Unused = {2 - 8} $1 - 2 = D(2) + W(2) = 1 + 2 = 3$. Are all new paths. |
| 2 | Insert 2 into Used set, Used = {1, 2} Shortest path $1 - 1 = D(1) + W(1) = 0 + 0 = 0$. ----- Newly chosen, $1 - 2 = D(2) + W(2) = 1 + 2 = 3$. Search from 2 | Unused = {3 - 8} $1 - 2 - 3 = D(3) + W(3) = 4 + 4 = 8$, $1 - 2 - 5 = D(5) + W(5) = 4 + 3 = 7$. Are all new paths. |
| 3 | Insert 3, 5 into Used set, Used = {1, 2, 3, 5} Shortest path $1 - 1 = D(1) + W(1) = 0 + 0 = 0$, $1 - 2 = D(2) + W(2) = 1 + 3 = 3$. ----- Newly chosen, $1 - 2 - 3 = D(3) + W(3) = 4 + 4 = 8$, $1 - 2 - 5 = D(5) + W(5) = 4 + 3 = 7$. Search from 3, 5 | Unused = {4, 6, 7, 8} $1 - 2 - 3 - 6 = D(6) + W(6) = 6 + 12 = 18$, $1 - 2 - 5 - 4 = D(4) + W(4) = 8 + 9 = 17$, $1 - 2 - 5 - 6 = D(6) + W(6) = 8 + 7 = 15$, $1 - 2 - 5 - 8 = D(8) + W(8) = 8 + 6 = 14$. Are all new paths. |
| 4 | Insert 6, 4, 8 into Used set, Used = {1, 2, 3, 5, 6, 4, 8} Shortest path $1 - 1 = D(1) + W(1) = 0 + 0 = 0$, $1 - 2 = D(2) + W(2) = 1 + 2 = 3$, $1 - 2 - 3 = D(3) + W(3) = 4 + 4 = 8$, $1 - 2 - 5 = D(5) + W(5) = 4 + 3 = 7$. ----- Newly chosen, $1 - 2 - 3 - 6 = D(6) + W(6) = 6 + 12 = 18$, $1 - 2 - 5 - 4 = D(4) + W(4) = 8 + 9 = 17$, $1 - 2 - 5 - 8 = D(8) + W(8) = 8 + 6 = 14$. Search from 6, 4, 8 | Unused = {7} $1 - 2 - 3 - 6 - 5 = D(5) + W(5) = 10 + 14 = 24$, $1 - 2 - 3 - 6 - 8 = D(8) + W(8) = 10 + 16 = 26$, $1 - 2 - 3 - 6 - 7 = D(7) + W(7) = 10 + 15 = 25$, $1 - 2 - 5 - 8 - 6 = D(6) + W(6) = 10 + 10 = 20$. Are all new paths. |
| 5 | Insert 7 into Used set, Used = {1, 2, 3, 5, 6, 4, 8, 7} Shortest path $1 - 1 = D(1) + W(1) = 0 + 0 = 0$, $1 - 2 = D(2) + W(2) = 1 + 2 = 3$, $1 - 2 - 3 = D(3) + W(3) = 4 + 4 = 8$, $1 - 2 - 5 = D(5) + W(5) = 4 + 3 = 7$, $1 - 2 - 3 - 6 = D(6) + W(6) = 6 + 12 = 18$, $1 - 2 - 5 - 4 = D(4) + W(4) = 8 + 9 = 17$, $1 - 2 - 5 - 8 = D(8) + W(8) = 8 + 6 = 14$. ----- Newly chosen, $1 - 2 - 3 - 6 - 7 = D(7) + W(7) = 10 + 15 = 25$. Search from 6, 4, 8 | Unused = { } All visited. |

Table 3.4.1. The Path Search Process of Our Improved Dijkstra Algorithm

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Analysis

The simulation model is developed by using Visual C++ 14 in Code::Blocks(IDE), and the OS is Windows 10. It has Intel CORE i5, 7th Gen Processor and RAM capacity is 4GB.

In order to test the performance of our Improved Dijkstra Algorithm, we tested the algorithm with a real-life road map scenario. The environment map we constructed is shown in Fig. 3.

Consider an undirected graph $G = [V, E]$ with 41 nodes and 54 edges. V is a set of nodes and E is a set of Edges. $D(u)$ denotes count of connected road in node u . $d(x)$ denotes the total number of connected roads passes through all intersections from the root node to node x . $W(u, v)$ denotes cost for edge $(u - v)$. $w(x)$ denotes the total calculated weights from source node to node x .

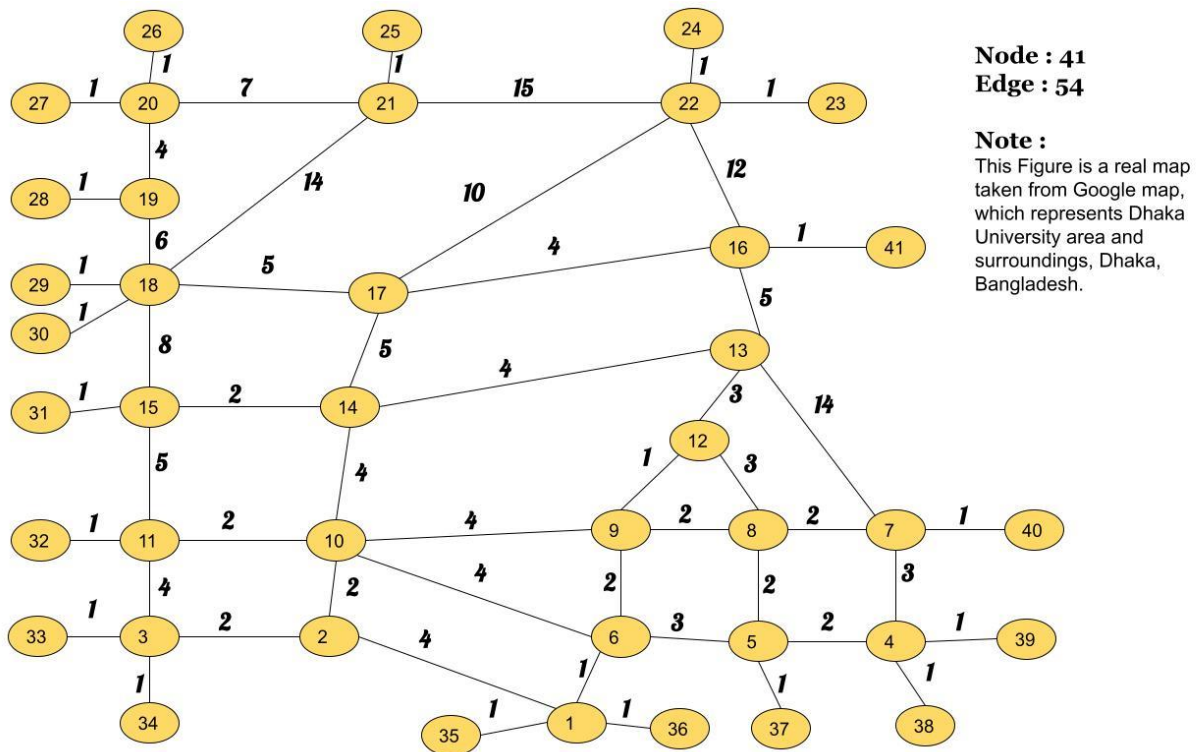


Figure 4.1.1. A real map taken from Google Map which represents Dhaka University area and surroundings, Dhaka, Bangladesh

Test-2:

Input: Taking 1 as source node, 7 as Goal Node.

Analysis: According to Fig 5.1, the shortest path for this input is $1 > 6 > 9 > 8 > 7$. found by Fig. 5.3 Output 2.

Here, According to Fig 5.3

$$1 - 6 = D(1) + W(1, 6) = 4 + 1 = 5$$

$$6 - 9 = D(6) + W(6, 9) = 4 + 2 = 6$$

$$9 - 8 = D(9) + W(9, 8) = 4 + 2 = 6$$

$$8 - 7 = D(8) + W(8, 7) = 4 + 2 = 6$$

So,

$$d(7) = 4 + 4 + 4 + 4 = 16$$

$$w(7) = 5 + 6 + 6 + 5 = 23$$

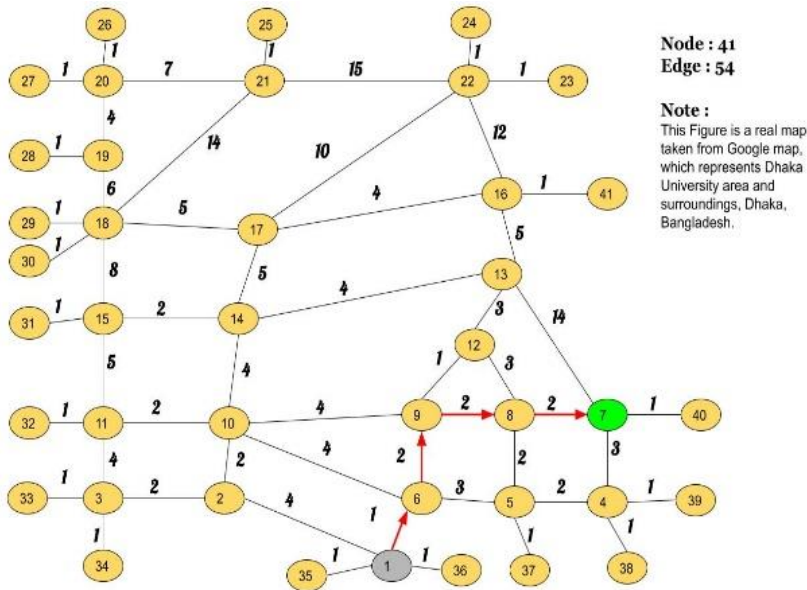


Fig. 5.3 Representing Test 2.

Test-3:

Input: Taking 4 as source node, 20 as Goal Node.

Analysis: According to Fig 5.1, the shortest path for this input is $4 > 5 > 6 > 10 > 14 > 17 > 18 > 19 > 20$. found by Fig. 5.4 Output 3.

Here, : According to Fig 5.4,

$$4 - 5 = D(4) + W(4, 5) = 4 + 2 = 6$$

$$5 - 6 = D(5) + W(5, 6) = 4 + 3 = 7$$

$$6 - 10 = D(6) + W(6, 10) = 4 + 4 = 8$$

$$10 - 14 = D(10) + W(10, 14) = 5 + 4 = 9$$

$$14 - 17 = D(14) + W(14, 17) = 4 + 5 = 9$$

$$17 - 18 = D(17) + W(17, 18) = 4 + 5 = 9$$

$$18 - 19 = D(18) + W(18, 19) = 6 + 6 = 12$$

$$19 - 20 = D(19) + W(19, 20) = 3 + 4 = 7$$

So,

$$d(20) = 4 + 4 + 4 + 5 + 4 + 4 + 6 + 3 = 34$$

$$w(20) = 6 + 7 + 8 + 9 + 9 + 9 + 12 + 7 = 67$$

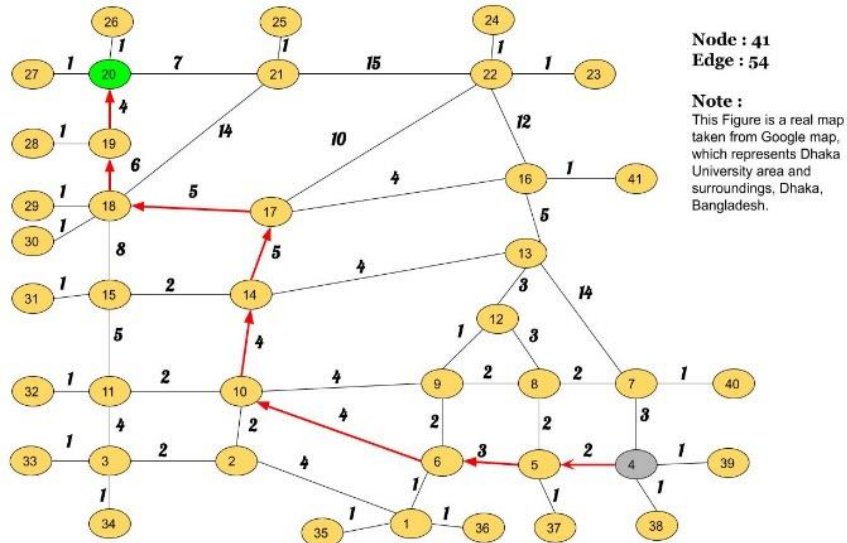


Fig. 5.4 Representing Test 3.

Now coming to the major part of our Algorithm. Suppose we want to calculate a path from a point to another point. And we have two different paths with equals weight of the path. In this situation we have to choose one of them. So, According to our problem we want to decrease the weight of the intersections. In this case, we have to take the path which have less weight of intersection.

For Example,

INPUT:

Source :13

Goal: 38

Path-1,

Path is: 13 > 7 > 4 > 38

Here,

$$13 - 7 = D(13) + W(13, 7) = 4 + 14 = 18$$

$$7 - 4 = D(7) + W(7, 4) = 4 + 3 = 7$$

$$4 - 38 = D(4) + W(4, 38) = 4 + 1 = 5$$

So,

$$d_1(38) = 4 + 4 + 4 = 12$$

$$w_1(38) = 18 + 7 + 5 = 30$$

Path-2,

Path is: 13 - 12 - 8 - 5 - 4 - 38

Here,

$$13 - 12 = D(13) + W(13, 12) = 4 + 3 = 7$$

$$12 - 8 = D(12) + W(12, 8) = 3 + 3 = 6$$

$$8 - 5 = D(8) + W(8, 5) = 4 + 2 = 6$$

$$5 - 4 = D(5) + W(5, 4) = 4 + 2 = 6$$

$$4 - 38 = D(4) + W(4, 38) = 4 + 1 = 5$$

So,

$$d_2(38) = 4 + 3 + 4 + 4 + 4 = 19$$

$$w_2(38) = 7 + 6 + 6 + 6 + 5 = 30$$

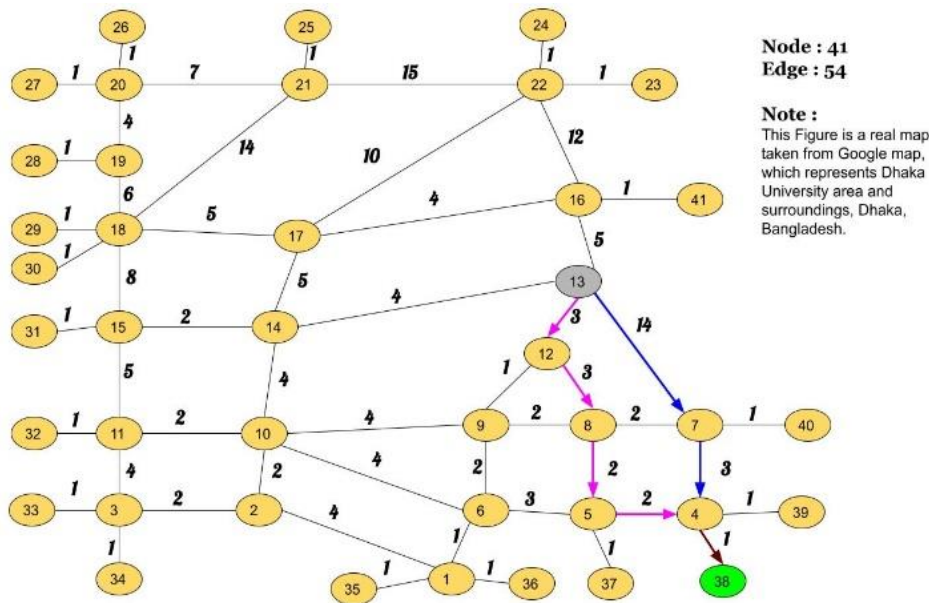


Fig. 5.5 Representing Special Case.

Following this two Path, we have two different paths for one input according to Fig-5.1. And we are taking Path-1. Though these two paths have equals weight which is $w_1(38) = 30$ and $w_2(38) = 30$. But the weights of intersections are not same. So, we can choose between them. Since Path-1 has less weight of intersections which is $d_1(38) = 12$. That means $d_1(38) = 12 < d_2(38) = 19$. That's why we are taking Path-1 for better results.

Chapter 5

CONCLUSION

In this paper we introduced an Improved Dijkstra algorithm to find out the shortest path. We consider intersections to calculate accurate weights for a path. We believe our proposed technique is very useful and realistic for finding a path. This algorithm can be applied in many applications. Our algorithm was implemented in C++ and used Code::Blocks to run this algorithm. In the future, we will try to solve more problems with the shortest path technique.

REFERENCES

- [1] Dijkstra E W. "A note on two problems in connexion with graphs," Numerische mathematik, vol. 1, no. 1, pp. 269-271, 1959.
- [2] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths," IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, 1968.
- [3] G. Qing, Z. Zheng, X. Yue "Path-planning of Automated Guided Vehicle based on Improved Dijkstra Algorithm" in Conf. 29th Chinese Control and Decision Conference, Beijing, china, 2017.
- [4] N. Makariye "Towards Shortest Path Computation using Dijkstra Algorithm" in Conf. 2017 International Conference on IoT and Application (ICIOT), Nagapattinam, India, 19-20 May 2017.
- [5] L. Wenzheng, L. Junjun, Y. Shunli. "An Improved Dijkstra's Algorithm for Shortest Path Planning on 2D Grid Maps" in Conf. 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), Beijing, China, 2019.
- [6] J. Liu, W. Li. "Aggressive Heuristic Search for Sub-Optimal Solution on Path Planning" in Conf. 2018 8th International Conference on Electronics Information and Emergency Communication (ICEIEC), Beijing, China, 2018.
- [7] S. Julius Fusic, P. Ramkumar, K. Hariharan. "Path planning of robot using modified Dijkstra Algorithm" in Conf. 2018 National Power Engineering Conference (NPEC), Madurai, India, 2018.
- [8] "Google Maps", "<https://www.google.com/maps/@23.7338497,90.3906813,17z>", 2021.

APPENDIX A

Pseudo code: Improved Dijkstra Algorithm

```
function Improved_Dijkstra(Graph, Degree, root)
  for each vertex v in Graph:
    dist[v] := infinity
    total_Deg[v] := 0
    prev[v] := undefined

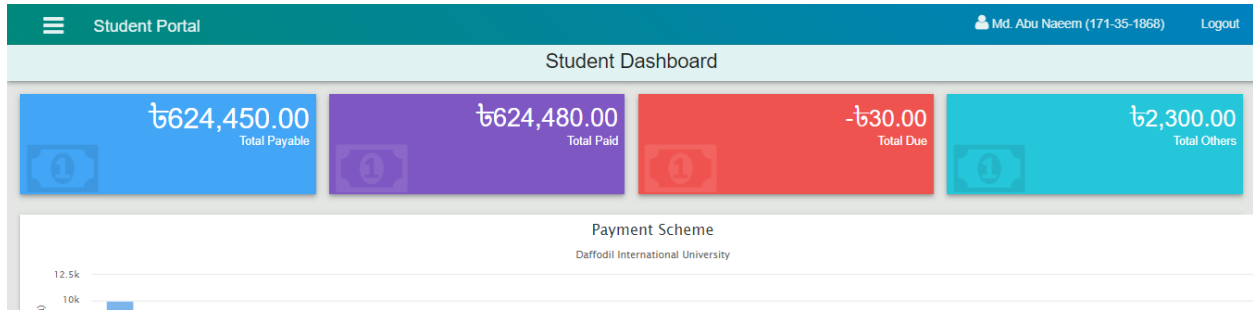
  dist[root] := 0
  PQ := add root to PQ

  while PQ != empty do
    u := node in PQ with smallest dist[]
    remove u from PQ
    for each adjacent v of u do
      current_deg = total_Deg[u] + Degree[u]
      current_dis = dist[u] + cost(u, v) + Degree[u]

      if current_dis < dist[v] Or (current_dis == dist[v] and current_deg < total_Deg[v]) then
        dist[v] := current_dis
        total_Deg[v] := current_deg
        prev[v] := u
        add v to PQ
      end if
    end for
  end while
  return {prev, dist}
end function
```

APPENDIX B

Accounts Clearance



The screenshot shows the 'Registration/Exam Clearance' section with a teal header containing 'Student Portal', user information 'Md. Abu Naeem (171-35-1868)', and a 'Logout' link. Below the header is a table with the following data:

| Semester | Registration | Mid Term Exam | Final Exam/Assessment |
|--------------|--------------|---------------|-----------------------|
| Spring, 2017 | ✓ | ✓ | ✓ (Final Exam) |
| Summer, 2017 | ✓ | ✓ | ✓ (Final Exam) |
| Fall, 2017 | ✓ | ✓ | ✓ (Final Exam) |
| Spring, 2018 | ✓ | ✓ | ✓ (Final Exam) |
| Summer, 2018 | ✓ | ✓ | ✓ (Final Exam) |
| Fall, 2018 | ✓ | ✓ | ✓ (Final Exam) |
| Spring, 2019 | ✓ | ✓ | ✓ (Final Exam) |
| Summer, 2019 | ✓ | ✓ | ✓ (Final Exam) |
| Fall, 2019 | ✓ | ✓ | ✓ (Final Exam) |
| Spring, 2020 | ✓ | ✓ | ✗ (Final Exam) |
| Fall, 2020 | ✓ | ✗ | ✓ (Final Exam) |