Modification of Toroslu's Improved Floyd-Warshall Algorithm Applied in Road Networks

Vincent Gerard Caro [1], *Adriane Kyle Cortez* [1], *Matthew Adrian Visto* [1], *Mark Christopher Blanco* [1], *Richard Regala* [1], *Vivien Agustin* [1]

[1]*Student, Computer Science Department, College of Engineering, Pamantasan ng Lungsod ng Maynila (University of the City of Manila), Intramuros, Manila 1002, Philippines.*

Corresponding Author: vgbcaro2018@plm.edu.ph

Abstract: - Travelers have been struggling to save time and resources due to worsening traffic. This problem with traffic has also been affecting the environment and economy. Shortest path algorithms aid in reducing the travel time of motorists. Toroslu's algorithm is a shortest path algorithm that returns the least cost between all pairs of vertices in a graph. In the interest of road networks and unpredictable traffic conditions, the algorithm used should cater for dynamic changes and have the ability to return the exact path. The researchers utilized the concept of predecessors and implemented Krishna, Moorthy, Gayathri's dynamic graph algorithm to address the limitation of Toroslu's algorithm. Results showed that the modified algorithm returns the correct costs and paths for every change made in the graph.

Key Words: - T*raffic, Shortest path algorithm, Dynamic graphs, Predecessors.*

I. Introduction

The continuous growth of population and number of vehicles have greatly contributed to problems with today's traffic [5] [10]. Furthermore, these problems also affect the environment and the economy. Effective utilization of technology can be used to improve the travel time of both motorists and commuters [13]. In relation to this, the development of shortest path algorithms has been vital in solving these problems [12] [2]. These algorithms are also applied in other scenarios such as city emergency handling and route planning in parking lots.
The shortest path problem has the objective of finding the shortest path between two nodes in a graph [13].

Manuscript revised June 07, 2023; accepted June 08, 2023. Date of publication June 09, 2023.

This paper available online at www.ijprse.com

ISSN (Online): 2582-7898; SJIF: 5.59

Finding an efficient algorithm for applications such as logistics and traffic routing would be beneficial to both the system and users [2] [16].
Floyd-Warshall's (FW) algorithm is an all-pair shortest path algorithm (APSP). APSPs are algorithms that find the cost between all pairs of vertices in a graph [6]. Unfortunately, FW has a run time of $O(n^3)$. The improvement of the algorithm of Toroslu [26] focuses on enhancing the original FW's time complexity. Based on their experiments, the improved algorithm performs better compared to Dijkstra's, Johnson's, and the original FW. However, since the algorithm is not dynamic and does not return an exact path, this paper aims to find a solution for these deficiencies.

II. Related Studies

Road networks are defined as the most basic transport infrastructure in urban areas [29]. These are composed of interconnected road segments that cater to vehicle and pedestrian traffic. This is modeled in a graph by using the intersections as nodes or vertices and using the road segments as edges [28]. An adjacency matrix can be used to represent road networks as directed graphs [15]. In relation to this, [13] stated that every traveler wants to make the most efficient use of their time and money. Utilizing technology makes it simple for them to find the shortest distance on the route.

Wang [28] defines the shortest path problem as a classical endeavor in graph theory. The idea is to find the shortest path between a source and destination node in a network. Shortest path algorithms are widely used in areas similar to logistics, train routes, and rescue operations [8] [25] [30]. Alam & Faruq [2] conducted a study about using Dijkstra's algorithm to solve for the shortest path in road networks. Their research stated that the algorithm can be used to find the shortest path between one city to every other city in the map.

Toroslu's algorithm is an improved version of Floyd-Warshall's algorithm. The following are the primary changes from the original Floyd-Warshall algorithm: instead of examining the entire matrix, the second and third loops simply investigate the incoming and outgoing edges of the vertex k. Thus, compared to the classic Floyd-Warshall technique, these loops would iterate much less for sparse graphs. In addition to this, instead of processing vertices in the sequence of 1 to N, the algorithm would choose the following vertex k at each iteration by considering the number of its incoming and outgoing edges [26].

Predecessor lists are used in graph theory for extracting the route found after an algorithm has been executed. Each node within a graph requires a pointer or a reference to its predecessor in order to find out the way back after finding the destination through backtracking the predecessor list [7]. An example of predecessors being used is within the study of [23], wherein every individual road and junction within OpenDrive stores its predecessors. This, in turn, makes the model efficient in generating the routing graph of the road network.

According to [24], in their study, today's networks are not just big, but also dynamic, with edges constantly being added and withdrawn. Additionally, a study by [4] stated that a dynamic graph algorithm setting is where a number of vertices are fixed, and a graph's edges are inserted or removed in each step. Krishna, Moorthy, Gayathri [17] mentioned in their study that majority of real-world graphs are constantly changing. They conducted their study to create a solution that avoids re-computation for continuously changing graphs.

III. Existing Algorithm

3.1 Overview

Toroslu's algorithm is an improved version of Floyd-Warshall's algorithm that works best with sparse graphs [26]. The modification eliminated all useless relaxation attempts by creating a list of the vertices incoming and outgoing edges. The modification also has an additional heuristic that improves the performance of the algorithm.

3.2 Limitations of Toroslu's Algorithm

Pathfinding algorithms generally return both the least cost and the exact path to traverse. Toroslu's algorithm only returns the least cost and not the exact path. Furthermore, considering the dynamic nature of road network conditions, the algorithm requires a re-run every time the graph is changed.

3.3 Pseudocode of Toroslu's Algorithm

```
1:  $N$: number of vertices
2:  $A[1..N, 1..N]$: adjacency matrix ($d_{ij}^0$)
3:  procedure TOROSLU ($N, A[1..N, 1..N]$)
4:      for $i: 1 \to N$ do
5:          for $j: 1 \to N$ do
6:              if $(i \neq j) \land (A[i,j] \neq \infty)$ then
7:                  $out[i] \leftarrow out[i] \cup \{j\}$
8:                  $in[j] \leftarrow in[j] \cup \{i\}$
9:              end if
10:         end for
11:     end for
12:     for $k: 1 \to N$ do
13:         $all_k \leftarrow \{1,2,\dots,N\}$
14:         while $all_k \neq \emptyset$ do
15:             $k' \leftarrow \min \{in[k'] \times out[k'] : \forall k' \in all_k\}$
16:             $all_k \leftarrow all_k - k'$
17:         end while
18:         for each $i \in in[k]$ do
19:             for each $j \in out[k]$ do
20:                 if $A[i,j] > A[i,k] + A[k,j]$ then
21:                     if $A[i,j] = \infty$ then
22:                         $out[i] \leftarrow out[i] \cup \{j\}$
23:                         $in[i] \leftarrow in[j] \cup \{i\}$
24:                     end if
25:                     $A[i,j] \leftarrow A[i,k] + A[k,j]$
26:                 end if
27:             end for
28:         end for
29:     end for
30: end procedure
```

IV. Modified Algorithm

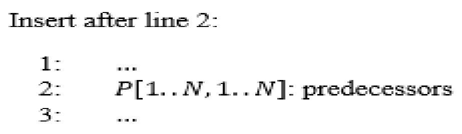
4.1 Modification of the Algorithm

To be able to return an exact path, the concept of predecessors was utilized. A 2D array is initialized before computing for the initial solution. Within Toroslu's algorithm, this is updated if a relaxation attempt is successful. Moving forward, the array is further updated when changes are made to the graph. To return the exact path for a given source and destination, traversal would start from the end node until it reaches the source node via manipulation of the said array.

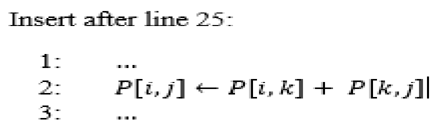
Next, a dynamic graph algorithm was incorporated into the existing algorithm. The dynamic graph algorithm used was the algorithm by Krishna, Moorthy, Gayathri, which is an APSP algorithm that utilizes data from an initial solution and a single-pair pathfinding algorithm. The algorithm has 3 sub-algorithms, namely: *edge insertion*, *edge deletion*, and *edge updation*. The modification is such that insertion can only be done on pairs of vertices where an edge does not exist, and deletion and updation can only be done on pairs of vertices where an edge exists. The researchers first implemented the dynamic algorithm to be usable with adjacency matrices and directed graphs. Afterwards, changes were made to the *weight updation* sub-algorithm while the other two re-used the same concept from the algorithm of Krishna, Moorthy, Gayathri. Continuous observation of the graph's properties led the researchers to conclude that some shortest paths are made up of smaller shortest paths. This concept was utilized to make the necessary changes to the *updation* sub-algorithm.

4.2 Pseudocode of the Modified Algorithm

The first modification done was the use of a predecessor array to return an exact path. This was done by adding the following steps in Toroslu's algorithm:


Insert after line 2:

```
...
$P[1..N, 1..N]$: predecessors
...
```


Insert after line 25:

```
...
$P[i,j] \leftarrow P[i,k] + P[k,j]$
...
```

Afterwards, Krishna, Moorthy, Gayathri's algorithm was modified to work using adjacency matrices. The idea for the *insertion* and *deletion* sub-algorithms was copied. Arrays *A[][]*, *L[][]*, and *P[][]* are the initial graph, the least cost of all pairs, and the predecessors of all pairs, respectively. Variables s_i and d_i are the source and destination of the newly inserted edge. The weight of the edge is represented via the variable *w*. As for the *deletion* part, the function *findpath* is used. This is a single-pair pathfinding algorithm that returns the least cost and the new predecessor of a given pair of vertices.

```
procedure EDGEINSERT ($A[1..N,1..N], L[1..N,1..N]$
                      $P[1..N,1..N], s_i, d_i, w_i$):
    $A[s_i, d_i] = w_i$
    for $i \rightarrow N$ do
        if $i \neq d_i \wedge L[i, s_i] \neq \infty$ do
            $w_{i \rightarrow d_i} \leftarrow L[i, s_i] + w_i$
            if $w_{i \rightarrow d_i} < L[i, d_i]$ do
                $L[i, d_i] = w_{i \rightarrow d_i}$
                $P[i, d_i] = s_i$
                for $j \rightarrow N$ do
                    if $i \neq j \wedge i \neq d_i \wedge L[d_i, j] \neq \infty$ do
                        $w_{i \rightarrow j} = w_{i \rightarrow d_i} + L[d_i, j]$
                        if $w_{i \rightarrow j} < L[i, j]$ do
                            $L[i, j] = w_{i \rightarrow j}$
                            $P[i, j] = P[d_i, j]$
                        end if
                    end if
                end for
            end if
        end if
    end for
end procedure
```

```
procedure EDGEDELETE ($A[1..N,1..N], P[1..N,1..N], s_i, d_i$):
    $A[s_i, d_i] = \infty$
    for $i \rightarrow N$ do
        for $j \rightarrow N$ do
            if $s_i$ and $d_i$ are adjacent within the path of $i$ and $j$ do
                $L[i, j], P[i, j] = \text{findpath}(A, i, j, s_i, d_i)$
            end if
        end for
    end for
end procedure
```

The *updation* sub-algorithm was further modified to also re-calculate for pairs whose shortest paths do not have the updated edge. The pseudocode for it is as follows:

```
procedure EDGEUPDATE ($A[1..N, 1..N]$, $L[1..N, 1..N]$,
                      $P[1..N, 1..N], s_i, d_i, w_i$):
    for $i$: $1 \rightarrow N$ do
        for $j$: $1 \rightarrow N$ do
            if $w_u < A[s_u, d_u]$ do
                if $s_i$ and $d_i$ are adjacent within the path of $i$ and $j$ do
                    $L[i,j] \leftarrow L[i,j] - (A[s_u, d_u] - w_u)$
                else
                    $w_n \leftarrow L[i, s_u] + w_u + L[d_u, j]$
                    if $w_n < L[i,j]$ do
                        $L[i,j] \leftarrow w_n$
                        if $d_u = j$ do
                            $P[i,j] = s_u$
                        else
                            $P[i,j] = P[d_u, j]$
                        end if-else
                    end if
                end if-else
            else
                if $s_i$ and $d_i$ are present within the path of $i$ and $j$ do
                    $A[s_u, d_u] = w_u$
                    $L[i,j], P[i,j] = \text{findpath}(A, i, j, s_i, d_i)$
                end if
            end if-else
        end for
    end for
    $A[s_u, d_u] = w_u$
end procedure
```

V. Methodology

An experimental design was used to determine if the modified algorithm accurately returns the least cost and the exact path for every state of the graph. The dataset used for testing the proposed algorithm was taken from the study of [27] about vehicle route planning. They used the road networks of five major cities that are represented via a graph, published by the Computer Science Department of The University of Utah. The researchers specifically used the graph of San Joaquin County's Road network for its ease of use. The network contains approximately 18,260 and 23,870 edges. Due to lack of resources, only a portion of the entire graph was used for this paper.

Dijkstra's algorithm was used as a benchmark to check for, if any, mistakes in the outputs. Accuracy was the performance metric used for the experiment.

VI. Results and Discussion

The following tables and figures represent the graphs used for testing and their corresponding solution. The graphs have nodes that represent intersections in the road network and directed edges with weights that illustrate the distance from the related nodes.

Looking at the outputs in Table I and Table II, the modified algorithm has successfully returned the least costs as well as the exact paths for all pairs in the graph from Fig. 1. The researchers also ran the problem graph through Dijkstra's algorithm, and it yielded the same result as the modified one.

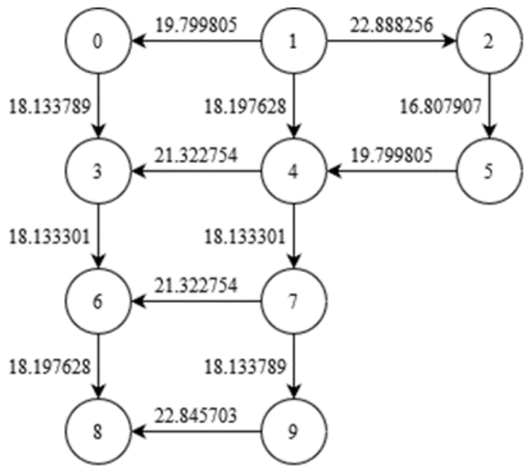


Fig.1. Initial Problem Graph

Table.1. Shortest Paths of Fig. 1 using the Modified Algorithm

SHORTEST PATHS			
0→3:	0 → 3	3→6:	3 → 6
0→6:	0 → 3 → 6	3→8:	3 → 6 → 8
0→8:	0 → 3 → 6 → 8	4→3:	4 → 3
1→0:	1 → 0	4→6:	4 → 7 → 6
1→2:	1 → 2	4→7:	4 → 7
1→3:	1 → 0 → 3	4→8:	4 → 7 → 6 → 8
1→4:	1 → 4	4→9:	4 → 7 → 9
1→5:	1 → 2 → 5	5→3:	5 → 4 → 3
1→6:	1 → 0 → 3 → 6	5→4:	5 → 4
1→7:	1 → 4 → 7	5→6:	5 → 4 → 7 → 6
1→8:	1 → 0 → 3 → 6 → 8	5→8:	5 → 4 → 7
1→9:	1 → 4 → 7 → 9	5→9:	5 → 4 → 7 → 6 → 8
2→3:	2 → 5 → 4 → 3	6→8:	5 → 4 → 7 → 9
2→4:	2 → 5 → 4	7→6:	6 → 8
2→5:	2 → 5	7→8:	7 → 6
2→6:	2 → 5 → 4 → 7 → 6	7→9:	7 → 6 → 8
2→7:	2 → 5 → 4 → 7	9→8:	7 → 9
2→8:	2 → 5 → 4 → 7 → 6 → 8	9→8:	9 → 8
2→9:	2 → 5 → 4 → 7 → 9		

Table.2. Least Costs of Fig. 1 using the Modified Algorithm

Modified Algorithm										
LEAST COST										
	0	1	2	3	4	5	6	7	8	9
0	0	X	X	18.133789	X	X	36.26709	X	55.464718	X
1	19.799805	0	22.888256	37.933594	18.197628	39.696163	56.066895	36.330929	75.264526	54.464718
2	X	X	0	57.930466	36.607712	16.807907	76.063766	54.741013	95.261398	72.874802
3	X	X	X	0	X	X	18.133301	X	37.330929	X
4	X	X	X	21.322754	0	X	39.456055	18.133301	58.653683	36.26709
5	X	X	X	41.122559	19.799805	0	59.255859	37.933105	78.453491	56.066895
6	X	X	X	X	X	X	0	X	19.197628	X
7	X	X	X	X	X	X	21.322754	0	40.520382	18.133789
8	X	X	X	X	X	X	X	X	0	X
9	X	X	X	X	X	X	X	X	22.845703	0

Looking at Fig.1, Table I, and Table II, nodes 2, 3, 4, and 5 are unreachable from node 7. Traversal to the mentioned nodes is now possible after the insertion of the new edge in Fig 2. The exact paths and least costs are also updated after the re-computation as seen in Table III and Table IV.

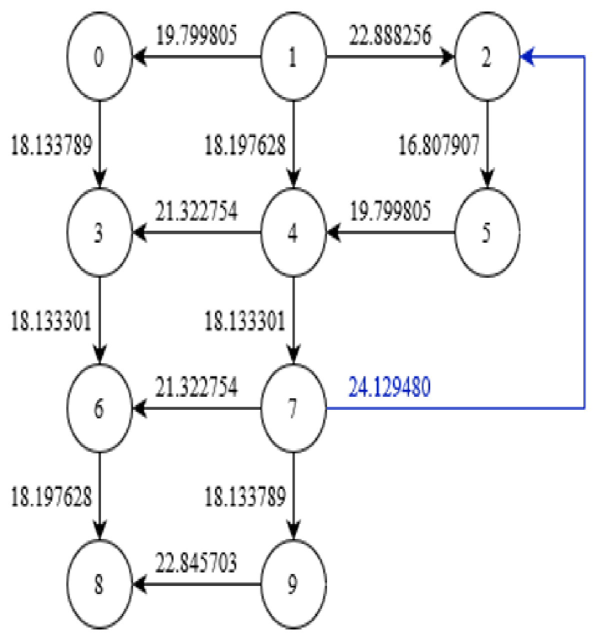


Fig.2. Problem Graph After Edge Insertion

Table.3. Shortest paths of Fig. 2 using the Modified Algorithm

SHORTEST PATHS			
0⟶3:	0 ⟶ 3	4⟶3:	4 ⟶ 3
0⟶6:	0 ⟶ 3 ⟶ 6	4⟶5:	4 ⟶ 7 ⟶ 2 ⟶ 5
0⟶8:	0 ⟶ 3 ⟶ 6 ⟶ 8	4⟶6:	4 ⟶ 7 ⟶ 6
1⟶0:	1 ⟶ 0	4⟶7:	4 ⟶ 7
1⟶2:	1 ⟶ 2	4⟶8:	4 ⟶ 7 ⟶ 6 ⟶ 8
1⟶3:	1 ⟶ 0 ⟶ 3	4⟶9:	4 ⟶ 7 ⟶ 9
1⟶4:	1 ⟶ 4	5⟶2:	5 ⟶ 4 ⟶ 7 ⟶ 2
1⟶5:	1 ⟶ 2 ⟶ 5	5⟶3:	5 ⟶ 4 ⟶ 3
1⟶6:	1 ⟶ 0 ⟶ 3 ⟶ 6	5⟶4:	5 ⟶ 4
1⟶7:	1 ⟶ 4 ⟶ 7	5⟶6:	5 ⟶ 4 ⟶ 7 ⟶ 6
1⟶8:	1 ⟶ 0 ⟶ 3 ⟶ 6 ⟶ 8	5⟶7:	5 ⟶ 4 ⟶ 7
1⟶9:	1 ⟶ 4 ⟶ 7 ⟶ 9	5⟶8:	5 ⟶ 4 ⟶ 7 ⟶ 6 ⟶ 8
2⟶3:	2 ⟶ 5 ⟶ 4 ⟶ 3	5⟶9:	5 ⟶ 4 ⟶ 7 ⟶ 9
2⟶4:	2 ⟶ 5 ⟶ 4	6⟶8:	6 ⟶ 8
2⟶5:	2 ⟶ 5	7⟶2:	7 ⟶ 2
2⟶6:	2 ⟶ 5 ⟶ 4 ⟶ 7 ⟶ 6	7⟶3:	7 ⟶ 2 ⟶ 5 ⟶ 4 ⟶ 3
2⟶7:	2 ⟶ 5 ⟶ 4 ⟶ 7	7⟶4:	7 ⟶ 2 ⟶ 5 ⟶ 4
2⟶8:	2 ⟶ 5 ⟶ 4 ⟶ 7 ⟶ 6 ⟶ 8	7⟶5:	7 ⟶ 2 ⟶ 5
2⟶9:	2 ⟶ 5 ⟶ 4 ⟶ 7 ⟶ 9	7⟶6:	7 ⟶ 6
3⟶6:	3 ⟶ 6	7⟶8:	7 ⟶ 6 ⟶ 8
3⟶8:	3 ⟶ 6 ⟶ 8	7⟶9:	7 ⟶ 9
4⟶2:	4 ⟶ 7 ⟶ 2	9⟶8:	9 ⟶ 8

Table.4. Least Costs of Fig. 2 using the Modified Algorithm

Modified Algorithm										
LEAST COST										
	0	1	2	3	4	5	6	7	8	9
0	0	X	X	18.133789	X	X	36.26709	X	55.464718	X
1	19.799805	0	22.888256	37.933594	18.197628	39.696163	56.066895	36.330929	75.264526	54.464718
2	X	X	0	57.930466	36.607712	16.807907	76.063766	54.741013	95.261398	72.874802
3	X	X	X	0	X	X	18.133301	X	37.330929	X
4	X	X	42.262779	21.322754	0	59.070686	39.456055	18.133301	58.653683	36.26709
5	X	X	62.062584	41.122559	19.799805	0	59.255859	37.933105	78.453491	56.066895
6	X	X	X	X	X	X	0	X	19.197628	X
7	X	X	24.12948	82.059944	60.73719	40.937386	21.322754	0	40.520382	18.133789
8	X	X	X	X	X	X	X	X	0	X
9	X	X	X	X	X	X	X	X	22.845703	0

After deleting the edge from node 1 to node 4, its shortest path is updated to 1⟶2⟶5⟶4 with a cost of 59.495968. The least cost and exact path are seen in Table V and Table VI.

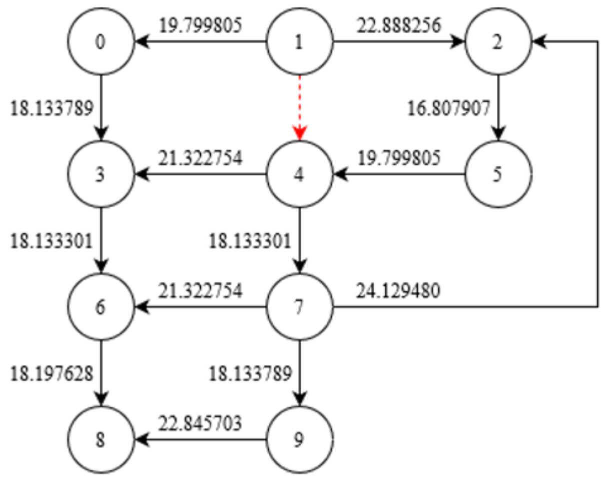


Fig.3. Problem Graph After Edge Insertion

Table.5. Shortest Paths of Fig. 3 using the Modified Algorithm

SHORTEST PATHS			
0→3:	0 → 3	4→3:	4 → 3
0→6:	0 → 3 → 6	4→5:	4 → 7 → 2 → 5
0→8:	0 → 3 → 6 → 8	4→6:	4 → 7 → 6
1→0:	1 → 0	4→7:	4 → 7
1→2:	1 → 2	4→8:	4 → 7 → 6 → 8
1→3:	1 → 0 → 3	4→9:	4 → 7 → 9
1→4:	1 → 2 → 5 → 4	5→2:	5 → 4 → 7 → 2
1→5:	1 → 2 → 5	5→3:	5 → 4 → 3
1→6:	1 → 0 → 3 → 6	5→4:	5 → 4
1→7:	1 → 2 → 5 → 4 → 7	5→6:	5 → 4 → 7 → 6
1→8:	1 → 0 → 3 → 6 → 8	5→7:	5 → 4 → 7
1→9:	1 → 2 → 5 → 4 → 7 → 9	5→8:	5 → 4 → 7 → 6 → 8
2→3:	2 → 5 → 4 → 3	5→9:	5 → 4 → 7 → 9
2→4:	2 → 5 → 4	6→8:	6 → 8
2→5:	2 → 5	7→2:	7 → 2
2→6:	2 → 5 → 4 → 7 → 6	7→3:	7 → 2 → 5 → 4 → 3
2→7:	2 → 5 → 4 → 7	7→4:	7 → 2 → 5 → 4
2→8:	2 → 5 → 4 → 7 → 6 → 8	7→5:	7 → 2 → 5
2→9:	2 → 5 → 4 → 7 → 9	7→6:	7 → 6
3→6:	3 → 6	7→8:	7 → 6 → 8
3→8:	3 → 6 → 8	7→9:	7 → 9
4→2:	4 → 7 → 2	9→8:	9 → 8

Table.6. Least Costs of Fig. 3 using the Modified Algorithm

Modified Algorithm										
LEAST COST										
	0	1	2	3	4	5	6	7	8	9
0	0	X	X	18.133789	X	X	36.26709	X	55.464718	X
1	19.799805	0	22.888256	37.933594	59.495968	39.696163	56.066895	77.629272	75.264526	95.763062
2	X	X	0	57.930466	36.607712	16.807907	76.063766	54.741013	95.261398	72.874802
3	X	X	X	0	X	X	18.133301	X	37.330929	X
4	X	X	42.262779	21.322754	0	59.070686	39.456055	18.133301	58.653683	36.26709
5	X	X	62.062584	41.122559	19.799805	0	59.255859	37.933105	78.453491	56.066895
6	X	X	X	X	X	X	0	X	19.197628	X
7	X	X	24.12948	82.059944	60.73719	40.937386	21.322754	0	40.520382	18.133789
8	X	X	X	X	X	X	X	X	0	X
9	X	X	X	X	X	X	X	X	22.845703	0

During the testing process, the researchers found a limitation with the approach of Krishna, Moorthy, Gayathri on their *updation* sub-algorithm. Looking at Fig. 4, edge [9,8] updated to a value of 11.21049 from a previous weight of 22.845703. Their algorithm states that only the shortest paths where the updated edge is present would require re-computations [17]. Upon observation of the shortest path of [7,8] and the updated edge, the shortest path should change to 7→9→8 from the previous 7→6→8. The comparison of results is displayed in Table VII and Table VIII.

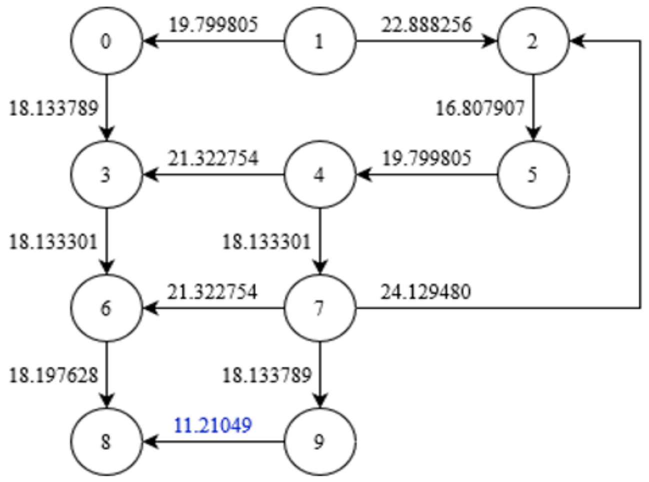


Fig.4. Problem Graph After Edge Updation

Table.7. Comparison of Shortest Paths of fig 4 Between Krishna, Moorthy, Gayathri's Algorithm and Modified Algorithm

SHORTEST PATHS			
Krishna, Moorthy, Gayathri's Algorithm		Modified Algorithm	
2→8:	2 → 5 → 4 → 7 → 6 → 8	2→8:	2 → 5 → 4 → 7 → 9 → 8
4→8:	4 → 7 → 6 → 8	4→8:	4 → 7 → 9 → 8
5→8:	5 → 4 → 7 → 6 → 8	5→8:	5 → 4 → 7 → 9 → 8
7→8:	7 → 6 → 8	7→8:	7 → 9 → 8

Table.8. Comparison of Shortest Paths of fig 4 Between Krishna, Moorthy, Gayathri's Algorithm and Modified Algorithm

LEAST COST						
	Krishna, Morrthy, Gayathri's Algorithm			Modified Algorithm		
	...	8	...	...	8	...
0	...	55.464718	...	...	55.464718	...
1	...	75.264526	...	...	75.264526	...
2	...	95.261398	...	...	84.085289	...
3	...	37.330929	...	...	37.330929	...
4	...	58.653683	...	...	47.477581	...
5	...	78.453491	...	...	67.277382	...
6	...	19.197628	...	...	19.197628	...
7	...	40.520382	...	...	29.34428	...
8	...	0	...	...	0	...
9	...	11.21049	...	...	11.21049	...

The experiments revealed that the modified algorithm provides a 100% accurate solution for all operations, including the *updation* sub-algorithm. When changes are made to the graph, the modifications do not require any re-use of Toroslu's algorithm to provide a solution. Lastly, the researchers were able to implement the algorithm of Krishna, Moorthy, Gayathri —originally created with tuples as its data type—to work with adjacency matrices.

VII. Conclusion

The study has successfully modified Toroslu's algorithm to return the exact path. The implementation of a dynamic feature was also accomplished. Krishna, Moorthy, Gayathri's algorithm has been modified to work with adjacency matrices and the limitation with the *updation* sub-algorithm was addressed. Experiments and results showed that the modified algorithm has an improved accuracy of 100%.

Acknowledgement:
The authors want to start by expressing their gratitude to God for His blessings and provision throughout this research as well as for guiding us through the challenges. He had a significant influence on our well-being and mental health as we conducted this research.
We would also like to express our sincere gratitude to Sir Mark Christopher Blanco, who served as our thesis adviser, for his direction and knowledge. His insightful advice and input were extremely helpful in guiding our study and improving its quality.
Additionally, Sir Richard Regala and Ma'am Vivien Agustin, deserve our gratitude for their thoughtful remarks, constructive criticism, and helpful suggestions that have significantly improved our work.
Lastly, we would like to thank our friends and family from the bottom of our hearts for their constant support and understanding during this difficult journey. Their unwavering faith in us has been a motivating source for conquering challenges and maintaining motivation.

References

[1]. AbuSalim, S. W., Ibrahim, R., Saringat, M. Z., Jamel, S., & Wahab, J. A. (2020, September). Comparative analysis between dijkstra and bellman-ford algorithms in shortest path optimization. In IOP Conference Series: Materials Science and Engineering (Vol. 917, No. 1, p. 012077). IOP Publishing.
[2]. Alam, M. A., & Faruq, M. O. (2019). Finding Shortest Path for Road Network Using Dijkstra's Algorithm. Bangladesh Journal of Multidisciplinary Scientific Research, 1(2), 41-45.
[3]. Ashong, D., Darkwah, K. F., & Tetteh, H. N. K. (2018). Comparison of Floyd-Warshall and Mills Algorithms for Solving all Pairs Shortest Path Problem. Case Study: Sunyani Municipality. International Journal of Physical Sciences Research, 2(1), 1-17.
[4]. Barenboim, L., & Maimon, T. (2019). Fully dynamic graph algorithms inspired by distributed computing: Deterministic maximal matching and edge coloring in sublinear update-time. Journal of Experimental Algorithmics (JEA), 24, 1-24.
[5]. Björck, E., & Omstedt, F. (2018). A comparison of algorithms used in traffic control systems.
[6]. Brodnik, A., & Grgurovič, M. (2022). Modifications of the Floyd-Warshall algorithm with nearly quadratic expected-time. Ars mathematica contemporanea, 22(1), P1-01.
[7]. Cao, Q. (2018). Exploiting Problem Structure in Pathfinding.
[8]. Dermawan, T. S. (2019). Comparison of dijkstra dan floyd-warshall algorithm to determine the best route of train. IJID (International Journal on Informatics for Development), 7(2), 54-58.
[9]. Elhoseny, M., Tharwat, A., & Hassanien, A. E. (2018). Bezier Curve Based Path Planning in a Dynamic Field using Modified Genetic Algorithm. Journal of Computational Science, 25, 339–350.
[10]. Navarro-Espinoza, A., López-Bonilla, O. R., García-Guerrero, E. E., Tlelo-Cuautle, E., López-Mancilla, D., Hernández-Mejía, C., & Inzunza-González, E. (2022). Traffic flow prediction for smart traffic lights using machine learning algorithms. Technologies, 10(1), 5.
[11]. Fitch, D. T., & Handy, S. L. (2020). Road environments and bicyclist route choice: The cases of Davis and San Francisco, CA. Journal of transport geography, 85, 102705.

[12]. Gbadamosi, O. A., & Aremu, D. R. (2020, March). Design of a Modified Dijkstra's Algorithm for finding alternate routes for shortest-path problems with huge costs. In 2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS) (pp. 1-6). IEEE.

[13]. Gunawan, W., & Susafa'ati, B. S. (2019). Implementation of Dijkstra's Algorithm in the Shortest Route.

[14]. Kumar, P. Entropy Maximization Problem in Network using Dijkstra's-Floyd Warshall Algorithm. Int. J. Comput. Appl, 181(37), 38-42.

[15]. J. Hu, C. Guo, B. Yang and C. S. Jensen, "Stochastic Weight Completion for Road Networks Using Graph Convolutional Networks," 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 2019, pp. 1274-1285.

[16]. Kirono, S., Arifianto, M. I., Putra, R. E., Musolch, A., & Setiadi, R. Graph-Based Modeling and Dijkstra Algorithm for Searching Vehicle Routes on Highways, International Journal of Mechanical Engineering and Technology 9(8), 2018, pp. 12731280.

[17]. Krishna, A., Moorthy, A. K., & Gayathri, R. G. (2018, August). Heuristic Based Ex-FTCD Algorithm for Incremental Path Finding in Dynamic Graphs. In 2018 International Conference on Data Science and Engineering (ICDSE) (pp. 1-6). IEEE.

[18]. Díez-Gutiérrez, M., & Babri, S. (2020). Explanatory variables underlying the route choice decisions of tourists: The case of Geiranger Fjord in Norway, Volume 141, pp. 398-409.

[19]. Ng, M. K., Chong, Y. W., Ko, K. M., Park, Y. H., & Leau, Y. B. (2020). Adaptive path finding algorithm in dynamic environment for warehouse robot. Neural Computing and Applications, 32, 13155-13171.

[20]. Ntakolia, C., & Iakovidis, D. K. (2021). A swarm intelligence graph-based pathfinding algorithm (SIGPA) for multi-objective route planning. Computers & Operations Research, 133, 105358.

[21]. Oladele, T. O., Adegun, A. A., Ogundokun, R. O., Okeyinka, A. E., & Ayeni, L. (2019). Application of Floyd-Warshall's algorithm in air freight service in Nigeria. Int J Eng Res Technol, 12, 2529-2535.

[22]. Leila Pasandi, Mehrnaz Hooshmand, Morteza Rahbar, (2021). Modified A* Algorithm integrated with ant colony optimization for multi-objective route-finding; case study: Yazd, Applied Soft Computing, Volume 113, Part A.

[23]. Poggenhans, F., & Janosovits, J. (2020, September). Pathfinding and Routing for Automated Driving in the Lanelet2 Map Framework. In 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC) (pp. 1-7). IEEE.

[24]. Riondato, M., & Upfal, E. (2018). ABRA. ACM Transactions on Knowledge Discovery from Data, 12(5), 1–38.

[25]. Rosita, Y. D., Rosyida, E. E., & Rudiyanto, M. A. (2019). Implementation of dijkstra algorithm and multi-criteria decision-making for optimal route distribution. Procedia Computer Science, 161, 378-385.

[26]. Toroslu, I. H. (2021). Improving The Floyd-Warshall All Pairs Shortest Paths Algorithm. arXiv preprint arXiv:2109.01872.

[27]. Udhan, P., Ganeshkar, A., Murugesan, P., Permani, A. R., Sanjeeva, S., & Deshpande, P. (2022). Vehicle Route Planning using Dynamically Weighted Dijkstra's Algorithm with Traffic Prediction.

[28]. Wang, X. Z. (2018, September). The comparison of three algorithms in shortest path issue. In Journal of Physics: Conference Series (Vol. 1087, No. 2, p. 022011). IOP Publishing.

[29]. Wu, N., Zhao, X. W., Wang, J., & Pan, D. (2020, August). Learning effective road network representation with hierarchical graph neural networks. In Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 6-14).

[30]. Zarrinpanjeh, N., Javan, F. D., Naji, A., Azadi, H., De Maeyer, P., & Witlox, F. (2020). Optimum path determination to facilitate fire station rescue missions using ant colony optimization algorithms (case study: City of Karaj). The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 43, 1285-1291.