COMPARATIVE ANALYSIS AND IMPLEMENTATION OF DIJKSTRA'S AND FLOYD'S ALGORITHMS FOR OPTIMAL TRANSPORTATION NETWORK OF MOTORCYCLE TOWN SERVICE

PRESENTED DURING NAMP CONFERENCE AT OMU-ARAN, NIGERIA, 6TH – 9TH NOV., 2018

Idrisu M., Evans P. O. and Nyor N.

[1,3]**Department of Mathematics, Federal University of Technology Minna.**

[2]**Department of Mathematics, federal Polytechnic Bida**

Abstract

- The paper aimed at presenting the results of a comparative analysis of the shortest path using Dijkstra's and floyd's Algorithms to find minimum links for commercial motorcycles that ply the routes in particular town. For the two algorithms, minimum paths were identified to ease good movement and reduce cost of fueling for the commercial motorcyclists. The results shows that floyd's is more effective in solving shortest path between all pairs while dijkstra is more effective between a given pairs. It was concluded that both algorithms are effective/efficient (because they provide the same solution) depending on what the problem solver wants

- **Keywords:** Dijksta's Algorithm, Floyd's, Algorithm, Optimal transportation Network, Town Service.

1.0 Introduction

Networks arise in numerous settings and in a variety of ways, ranging from transportation, electrical, and communication networks which pervade our daily lives. It is necessary for the movement of people, transportation of goods, communicate information and control of the flow of matter and energy. Network application is quite vast. Phenomena that are represented and analyzed as networks are roads, railways, cables, pipelines, production, distribution, project planning, facilities location,resource management, and financial planning, etc. It provides a powerful visual and conceptual aid that is used to portray the relationships between the components of systems in virtually every field of scientific, social, and economic endeavor.

1.0 Introduction

The shortest path problem is a problem of finding the shortest path or route from a starting point to a final destination. Generally, in order to represent the shortest path problem we use graphs. A graph is a mathematical abstraction that is useful for solving different networking problems. It is used to solve such problems contains sets of vertices and edges. Pairs of vertices are connected by edges, while movement from one vertex to other vertices can be done along the edges. It can be directed or undirected depending on the movement along the edges, either walking on both sides or on only one side. Lengths of edges are often called weights which are normally used to calculate the shortest path from a particular point to another point.

1.0 Introduction

In the real world, the graph theory can be applied to different scenarios. A practical example is map representation using a graph, where vertices and edges represent cities and routes that connect the cities respectively. One-way routes are directed graphs, while routes that are not one-way are undirected. Finding the shortest paths plays an important role in solving network based systems. In graph theory, a number of algorithms can be applied for finding the shortest path in a graph based network system. This reduces the complexity of the network path, the cost, and the time to build and maintain the network based systems.

1.0 Introduction

In recent times, planning efficient routes is very essential for business and industry with applications as varied as product distribution. It is essential for products or services to be delivered on time at the best price using the shortest available route. The shortest route network model is an efficient route that can be used in planning. This network model is applied in telecommunications and transportation planning. There are different types of algorithms that are used to solve shortest path problems. However, the Dijkstra’s and floyd’s algorithms are the concern for this paper.

2.0 Aim of the Study

This paper aimed to compare the performance of Dijkstra's and Floyd's algorithms in determining the minimum link paths and associated trip volumes between given origin-destination zones for commercial Motorcycles that carry out town service in Bida.

3.0 LITERATURE REVIEW

3.1 Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm named after its developer, Dijkstra Oliver in 1959. He conceived a graph search algorithm that can be used to solve the single-source shortest path problem for any graph that has a non-negative edge path cost. This graph search algorithm was later modified by Lee in 2006 and was applied to the vehicle guidance system. Dijkstra's algorithm or variations of it are the most commonly used route finding algorithm for solving the shortest path (Sadeghi-Niaraki*et al*., 2011). This vehicle guidance system is divided into two paths; namely, the shortest path and the fastest path algorithms(Chen et al., 2009).

3.1 Dijkstra's Algorithm

While the shortest path algorithm focuses on route length parameter and calculates the shortest route between each OD pair, the fastest path algorithm focuses on the path with minimum travel time. The future travel time can be predicted based on prediction models using historical data for link travel time information which can be daily, weekly or even a session.

Meghanathan (2012) reviewed Dijkstra's algorithm and Bellman-Ford algorithm for finding the shortest path in a graph. He concluded that the time complexity of Dijkstra's algorithm is O ($|E|*log\ |V|$) while the time complexity of the Bellman-Ford algorithm is O ($|V||E|$).

3.2 Floyd's Algorithm

Floyd-Warshall algorithm was familiarized by Robert Floyd in 1962 to resolve the problem of all pairs shortest path for weighted graph. However, it is essentially the same as algorithms previously published by Bernard Roy in 1959 and also by Stephen Warshall in 1962 for finding the transitive closure of a graph. It is a graph analysis algorithm for finding shortest paths in a weighted graph with positive or negative edge weights but with no negative cycles, and also for finding transitive closure of a relation R. A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves.

3.2 Floyd’s Algorithm

Borgwardt et al (2005)researched on the shortest-path kernel as applicable to all graphs on which Floyd-Warshall can be performed. The requirement of Floyd-Warshall is that cycles that have negative weight do not exist. This condition generally holds if distances are represented by edge labels.

4.0 Methodology

This work had its case study at Bida town which is the second largest city in Niger State with an estimated population of 178,840 (Census, 2007). It is located southwest of Minna, capital of Niger State. It is a dry, arid town. The major ethnic group is the Nupe. Bida is the headquarters of the Nupe Kingdom led by the Etsu Nupe.

Primary data was obtained by noting the time (in minutes) that it took motorcycles to ride from a particular origin to a particular destination; and counting the volume of flow between the eight selected zones. The data analysis was the application of Dijkstra's algorithm with the following steps:

i. Initialize by assigning a permanent label of zero to the home node (source). All other node labels are declared to be temporary and are equal to the direct distance from the source node to that node. Select the minimum of these temporary labels and declare it permanent. In the event of a tie, choose them arbitrarily.

ii. Suppose that node K has been assigned a permanent label most recently, consider the remaining nodes with temporary labels. Compare, one at a time, the temporary label of each node to the sum of the permanent label of node K and the direct distance from node K to the node under consideration. Assign the minimum of these distances as the new temporary label for that node. If the old temporary label is still minimal, then it will remain unchanged during this step.

iii. Select the minimum of all of the temporary labels and declare it permanent. In the case of ties, select just one of them and declare it permanent. If this happens to the destination node, then terminate. Otherwise go to step (i).

Floyd's Algorithm on the other hand represents an *n-node* network as a square matrix with *n* rows and *n* columns. It works by updating two matrices, namely and, n times for an *n* - node network. The matrix, in any iteration *k*, gives the value of the shortest distance (time) between all pairs of nodes (*i*, *j*) as obtained till the k^{th} iteration. The matrix has as its elements. The value of gives the immediate predecessor node from node *i* to node *j* on the shortest path as determined by the k^{th} iteration.

D_0 and S_0 give the starting matrices and D_n and S_n give the final matrices for an *n*-node system. The first task is to determine D_o and S_o. D_o is taken up first. The element d_{ij} of matrix D_o are defined as follows: If a link (branch) exists between nodes *i* and *j* the length of the shortest path between these nodes equals length *l (i, j)* of branch *(i, j)* which connects them. Should there be several branches between nodes *i* and node *j*, the length of the shortest path d_{ij} must equal the length of the shortest branch, that is,

$$d^0_{ij} = \min[l_1(i,j), l_2(i,j), \ldots l_m(i,j)] \qquad (1)$$

Where *m* is the number of branches between node *i* and node *j*.

It is clear that $d^0_{ij} = 0$ when $i = j$. In the case when there is no direct link between node i and node j, we have no information at the beginning concerning the length of the shortest path between these two nodes so we treat them as though they were infinitely far from each other, that is,

$$d^0_{ij} = \infty \qquad (2)$$

Elements s^0_{ij} of the predecessor matrix S_o are defined as follows:

First, we assume that $s^0_{ij} = i$, for i j, i.e. that for every pair of nodes (i, j) for i j, the immediate predecessor of node j on the shortest path leading from node i to node j is actually node i. After defining D_o and S_o the following steps are used repeatedly to determine D_n and S_n.

Step 1: Let $k = 1$

Step 2: We calculate elements d^{k}_{ij} of the shortest path length matrix found after the k^{th} passage through algorithm D_k using the following equation:

$$d^{k}_{ij} = \min[d^{k-1}_{ij}, d^{k-1}_{ik} + d^{k-1}_{kj}] \quad (3)$$

Step 3: Elements of predecessor matrix S_k found after the k^{th} passage through the algorithm are calculated as follows:

$$s^{k}_{ij} = \frac{s^{k-1}_{kj}, for\, d^{k}_{ij} \neq d^{k-1}_{ij}}{s^{k-1}_{ij}, otherwise} \quad (4)$$

Step 4: If $k = n$, the algorithm is finished. If $k < n$, increase k by 1, i.e. $K = k+1$ and return to step 2.

5.0 Data Presentation

The Data on commercial motorcycles popularly called Byke or Okada or Kabukabu in Bida were collected in June, 2018 from 8 Traffic Analysis Zones define as follows:

Table 1: Names of Zones and Nodes Identification

1	CIRICO JUNCTION	2	ESSO JUNCTION
3	BANGAYE	4	LUNCHITA
5	WADATA	6	POLY JUNCTION
7	BANYAGI	8	FEDERAL POLY BIDA

Table 2: Link Array Table

From/To	1	2	3	4	5	6	7	8
1	-	2	4	-	-	-	-	-
2	2	-	-	2	4	-	-	-
3	4	-	-	5	-	-	8	-
4	-	2	5	-	4	5	-	-
5	-	4	-	4	-	7	-	-
6	-	-	-	4	7	-	2	5
7	-	-	8	-	-	2	-	4
8	-	-	-	-	-	5	4	-

Source: The data were collected primarily by obtaining how much time (in minutes) it took Motorcycles between O-D zones

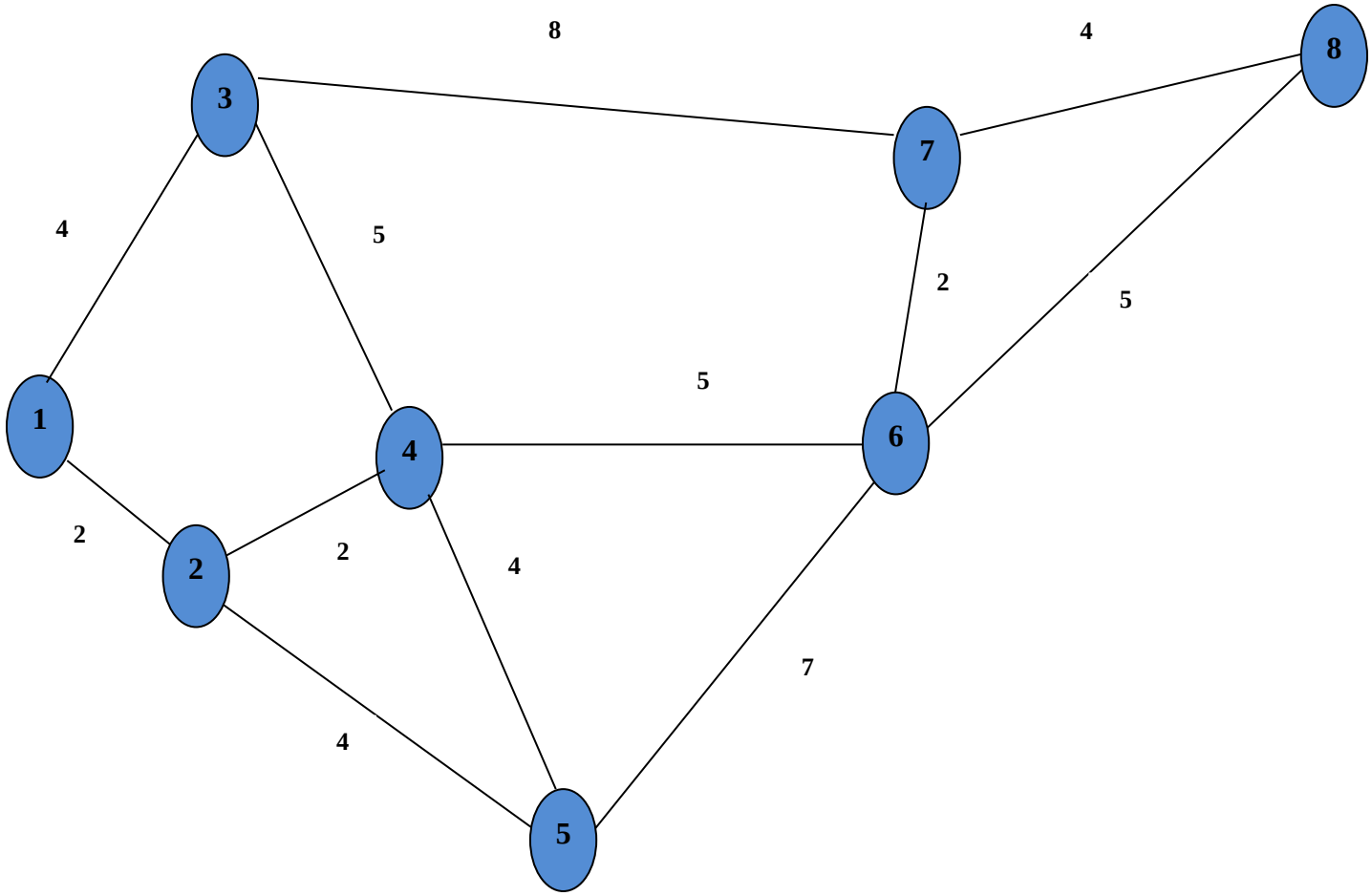

Figure 1: Network Diagram of Study Zones

6.0 Data Analysis

6.1 Dijkstra's Algorithm

Label node 1 with (0*,-) and declare it to be permanently labeled using a (*). Label the other nodes with upper bounds of (∞,-) and declare them to be temporary labels.

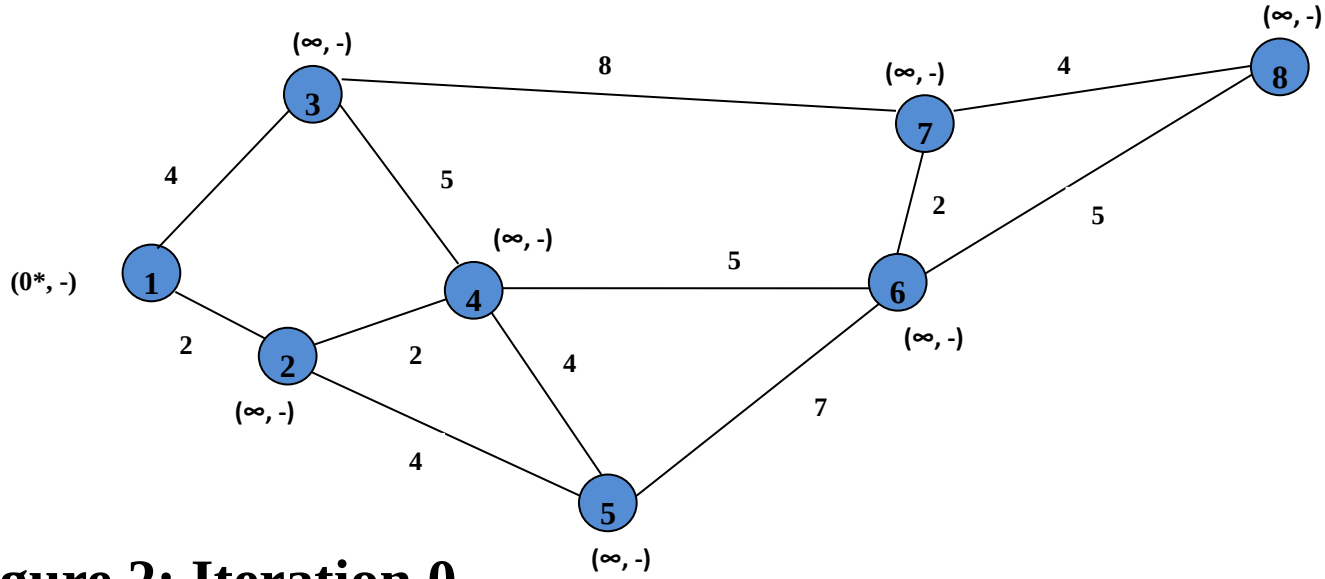

Figure 2: Iteration 0

Node 2 and node 3 can be reached from (the last permanent labeled) node 1, thus the list of labeled nodes becomes (temporary and permanent). We consider Node 2: min(∞, 0+1) = 1, Node 3: min(∞, 0+4) = 4

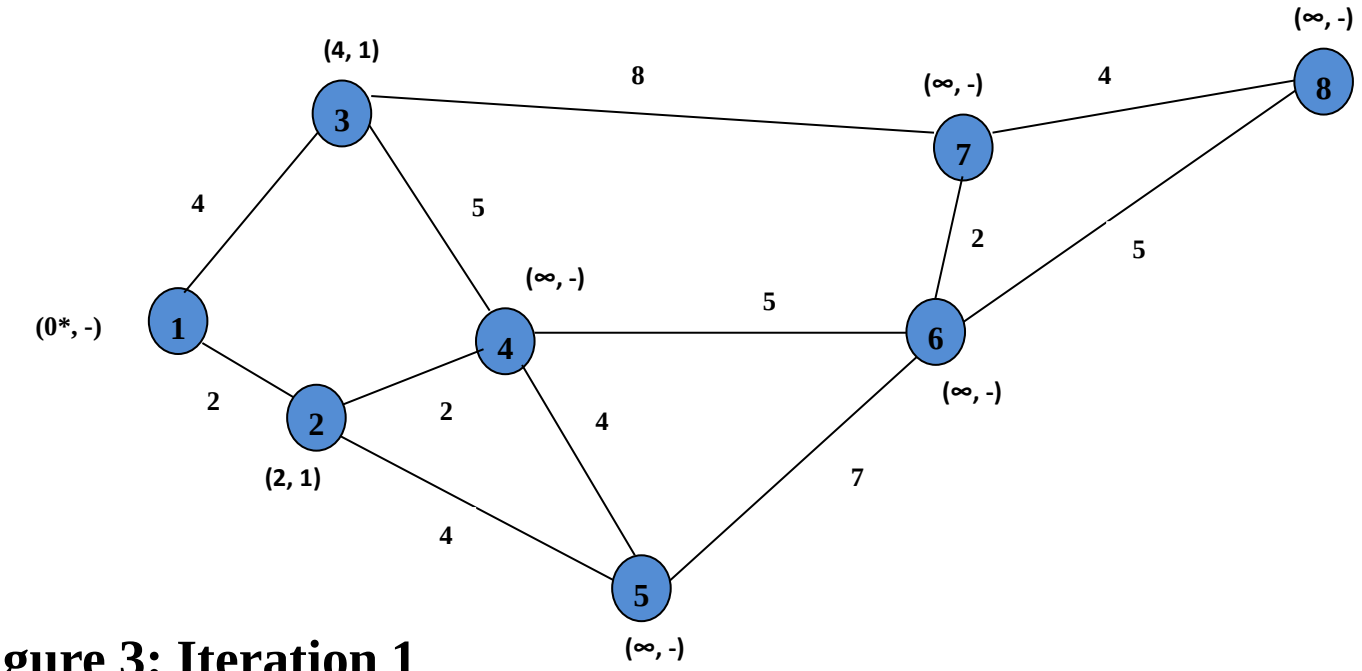

Figure 3: Iteration 1

For the two temporary labels, node 2 yields the smaller travel time (*2*). Thus the status of node 2 is changed to permanent.

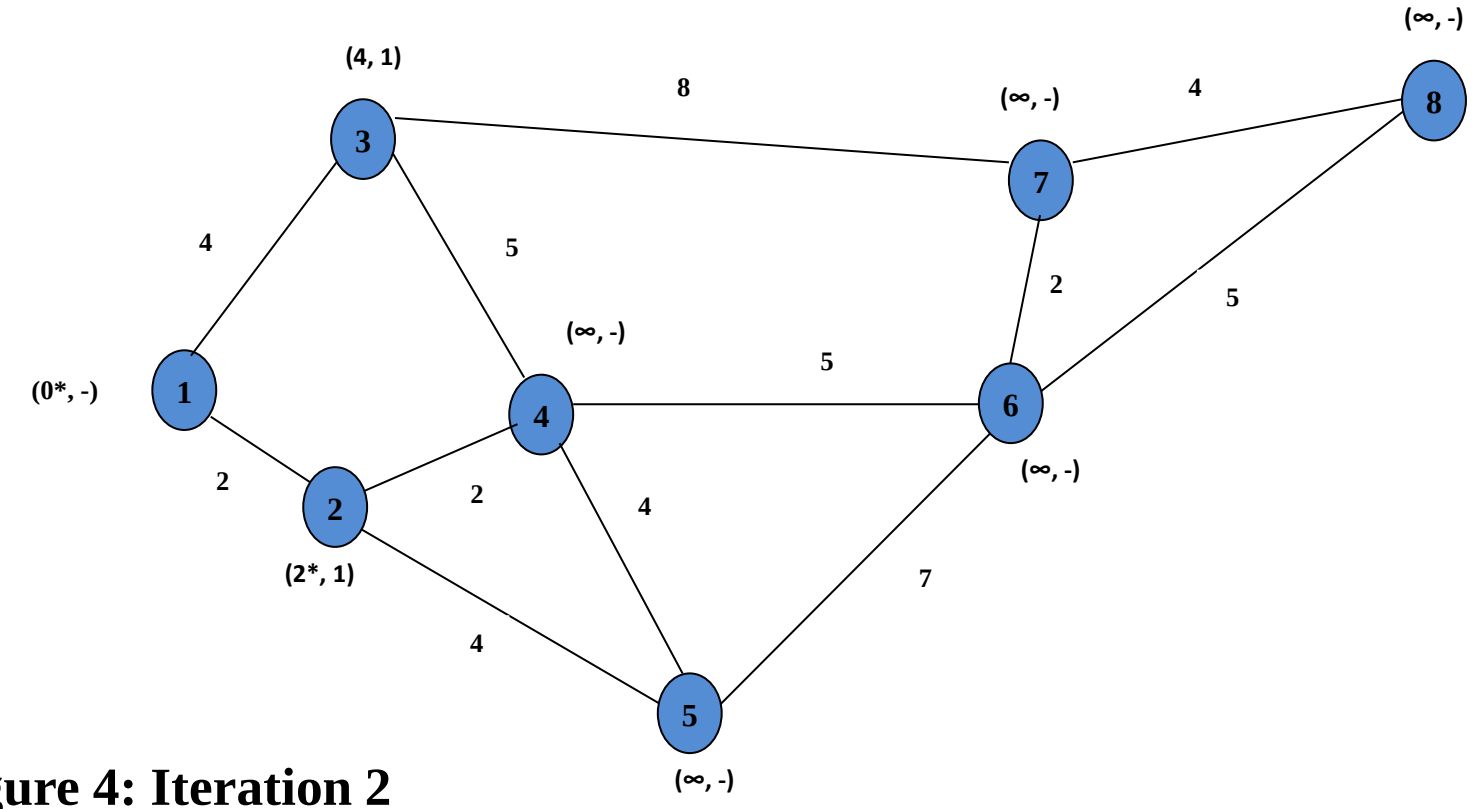

Figure 4: Iteration 2

Node 3 temporary label [4, 1] obtained in iteration 1 remains the same. We observed that node 3 and 4 can be reached through node 2. We consider Node3: min(∞, 0+4) = 4, Node 4: min(∞, 2+2) = 4 and Node 5: min(∞, 2+4) = 6.

From the temporally nodes we observed that node 3 and 4 are the smallest and break the tie arbitrary by taking node 3 and permanent it.

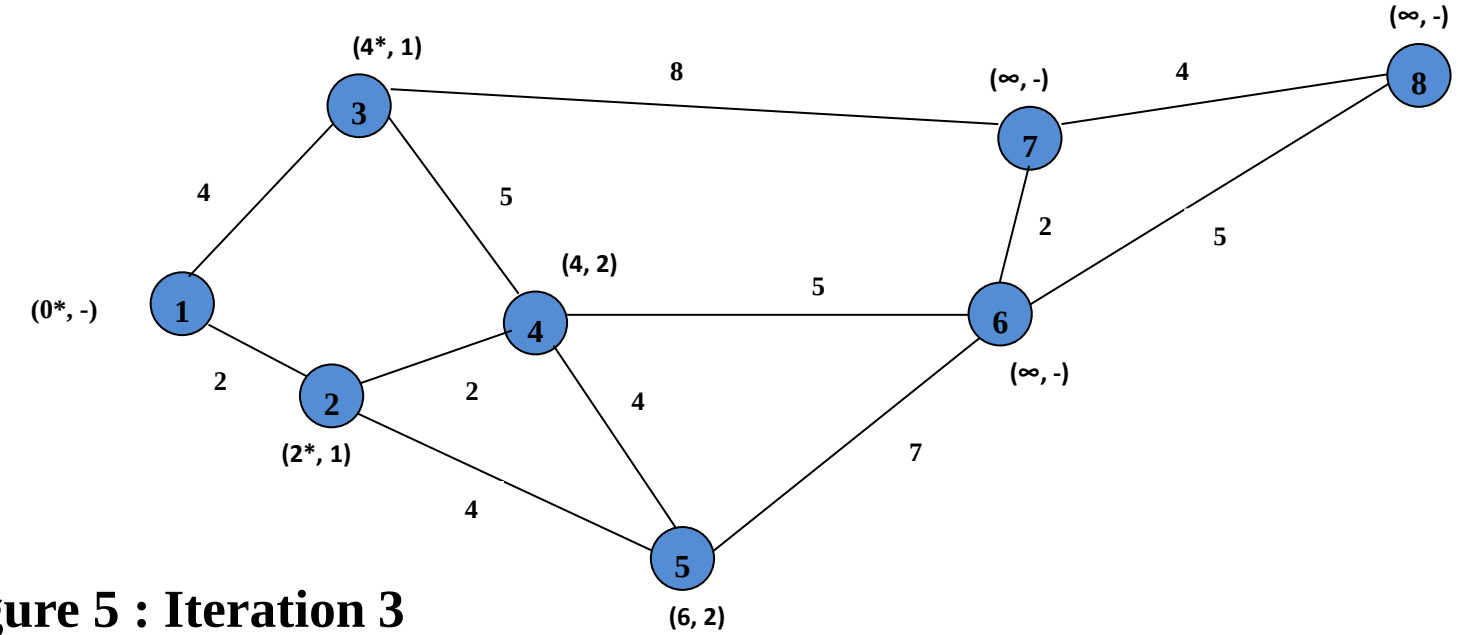

Figure 5 : Iteration 3

Node 4 and 5 remain temporary labeled and node 4 can again be reached through node 3. Similarly, node 7 can be reach through node 3. We consider Node 4: min(4, 4 + 5) = 4, Node 5: min(∞, 2+4) = 6 and Node 7: min(∞, 4+8) = 12.

From the temporary nodes we observed that node 4 the smallest and labeled it permanent.

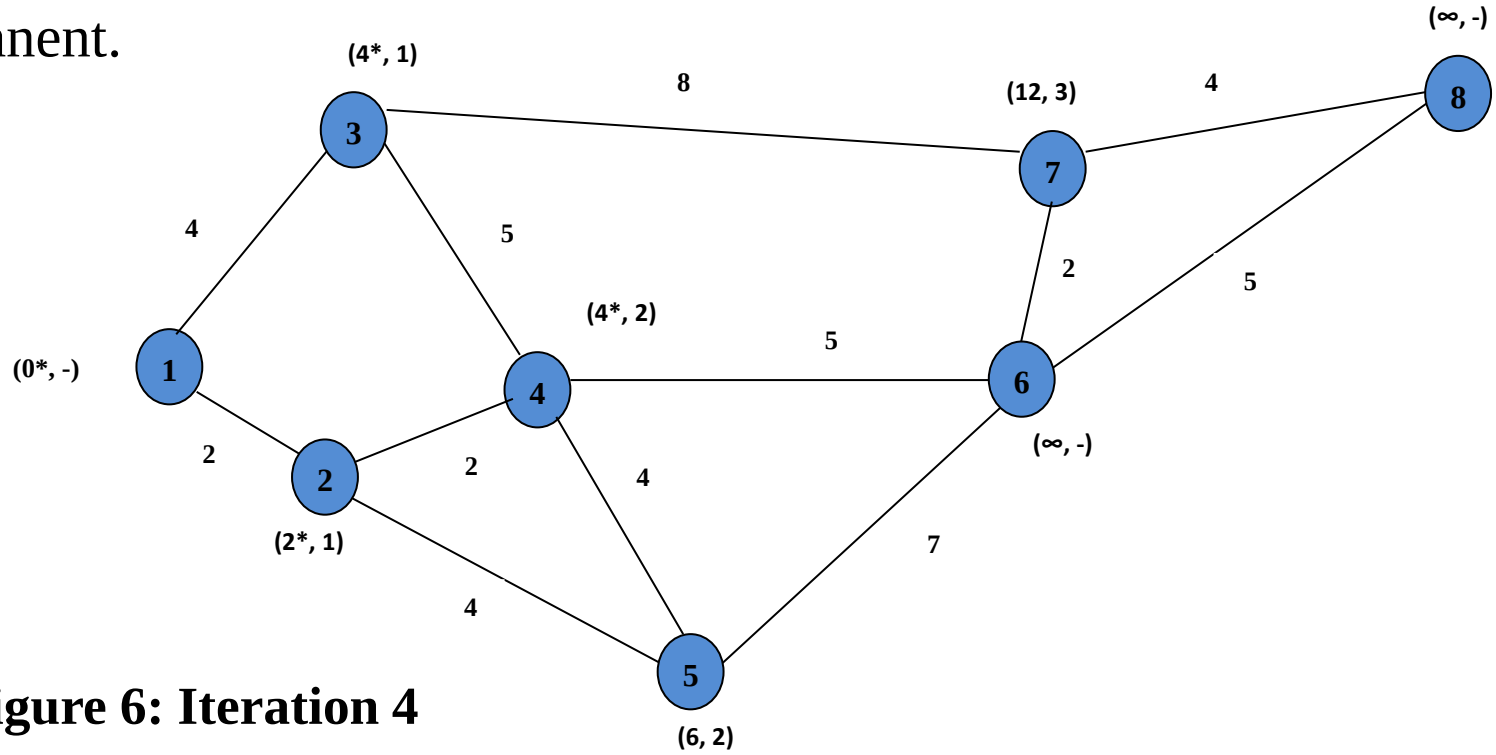

Figure 6: Iteration 4

Node 5 and 7 remain temporary and we observed that node 5 can again be reached through node 4. Similarly, node 6 can be reach through node 4. We consider Node 5: min(6, 4 + 4) = 6, Node 6: min(∞, 4+5) = 9 and Node 7: min(∞, 4+8) = 12

From the temporally nodes we observed that node 5 the smallest and labeled it permanent.

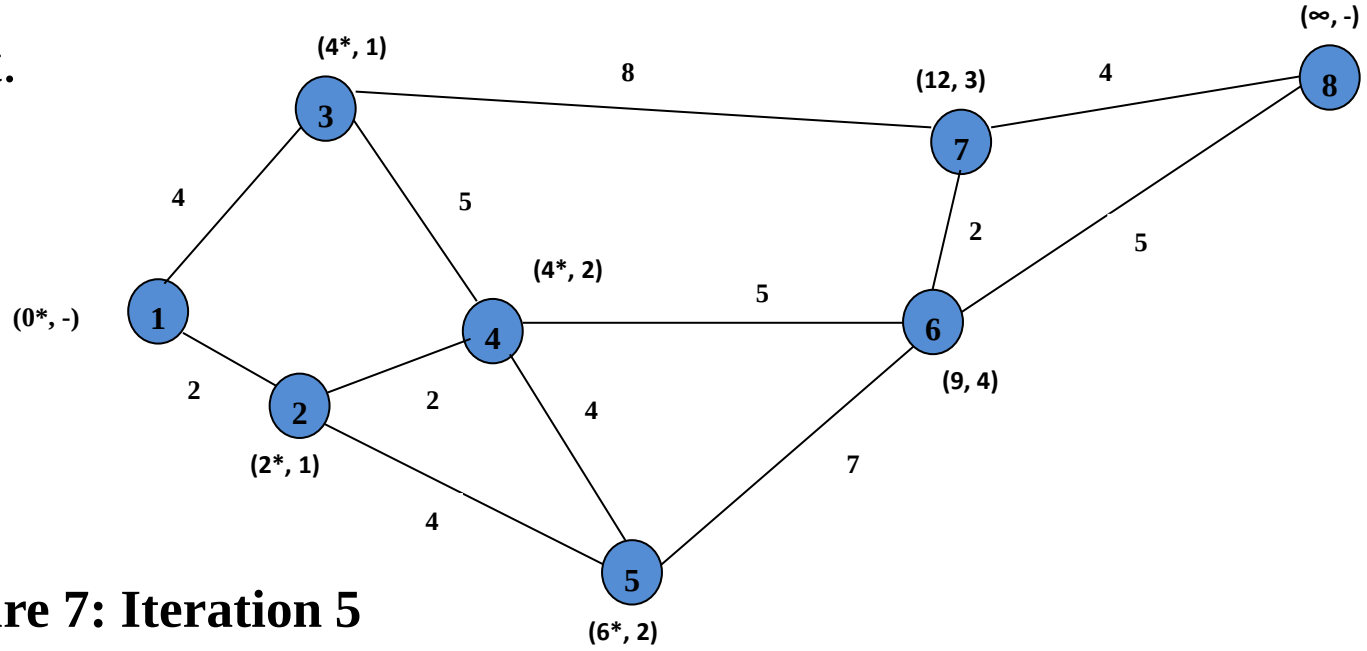

Figure 7: Iteration 5

Node 6 and 7 remain temporary and we observed that node 6 can again be reached through node 5. Similarly, node 6 can be reach through node 4. We consider Node 6: min(9, 6 + 7) = 9 and Node 7: min(∞, 4+8) = 12

From the temporally nodes we observed that node 6 the smallest and labeled it permanent.

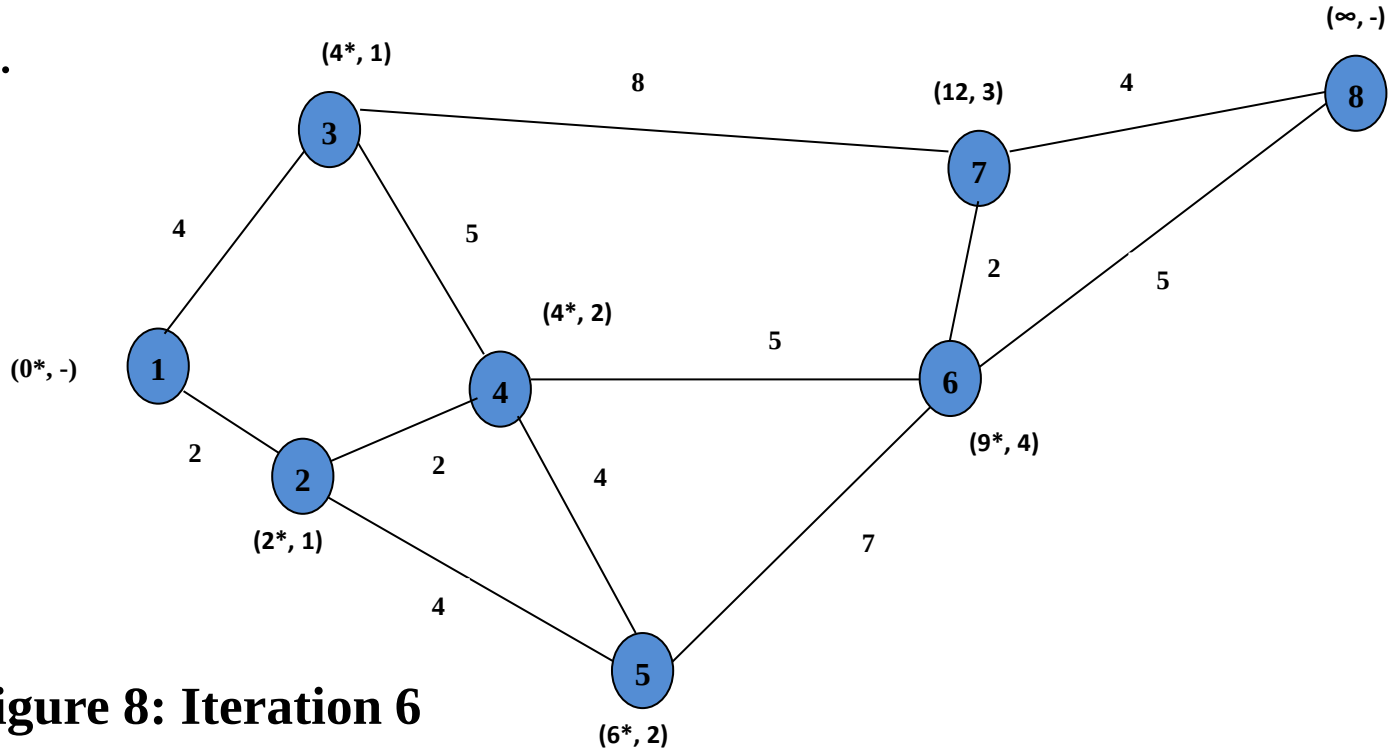

Figure 8: Iteration 6

Node 7 remains temporary and observed that node 7 can again be reached through node 6. Similarly, node 8 can be reach through node 6. We consider Node 7: min(12, 9 + 2) = 11 and Node 8: min(∞, 9 + 5) = 14

From the temporary nodes we observed that node 7 the smallest and labeled it permanent.

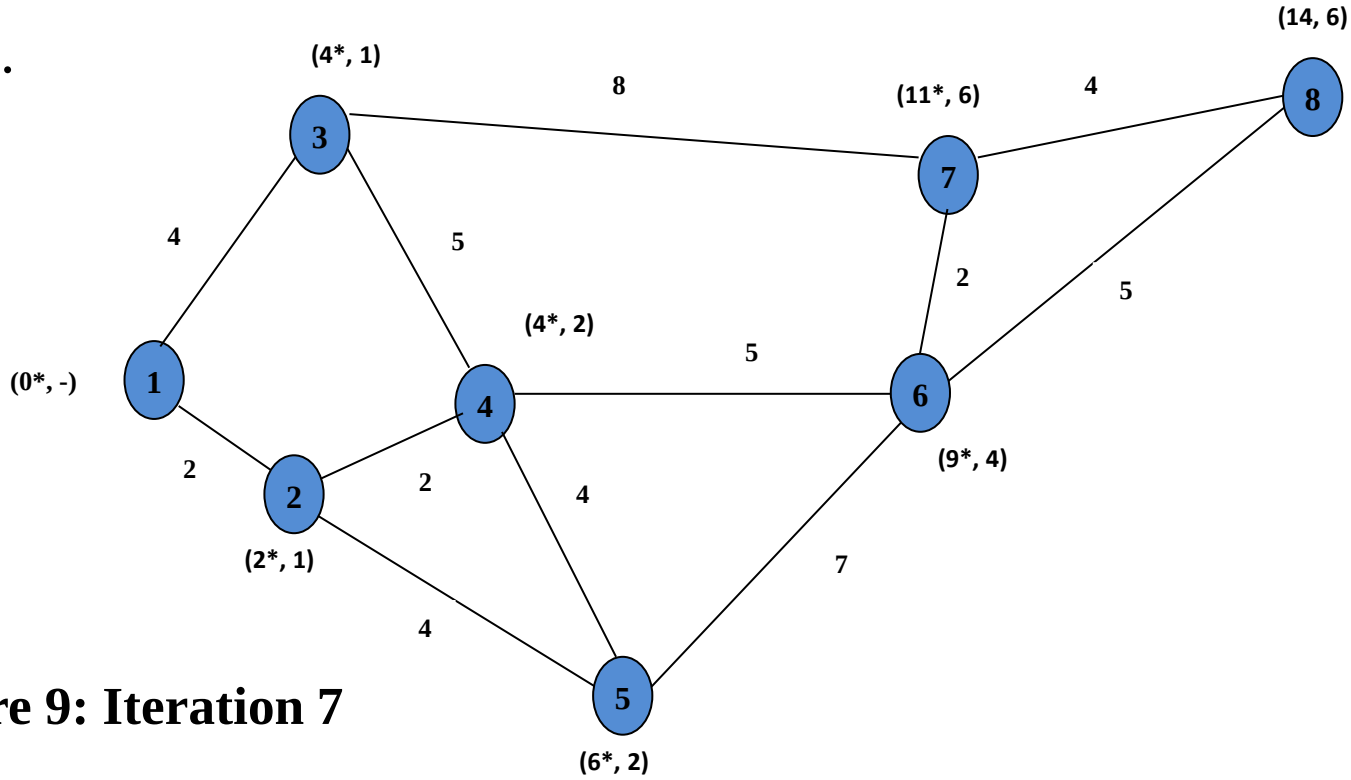

Figure 9: Iteration 7

From node 7, only node 8 can be reached so we check this: min (14, 11+ 4) = 14 (no change to the label).

The only temporary label remaining is that for node 8 so we label it permanently, and now the algorithm is complete.

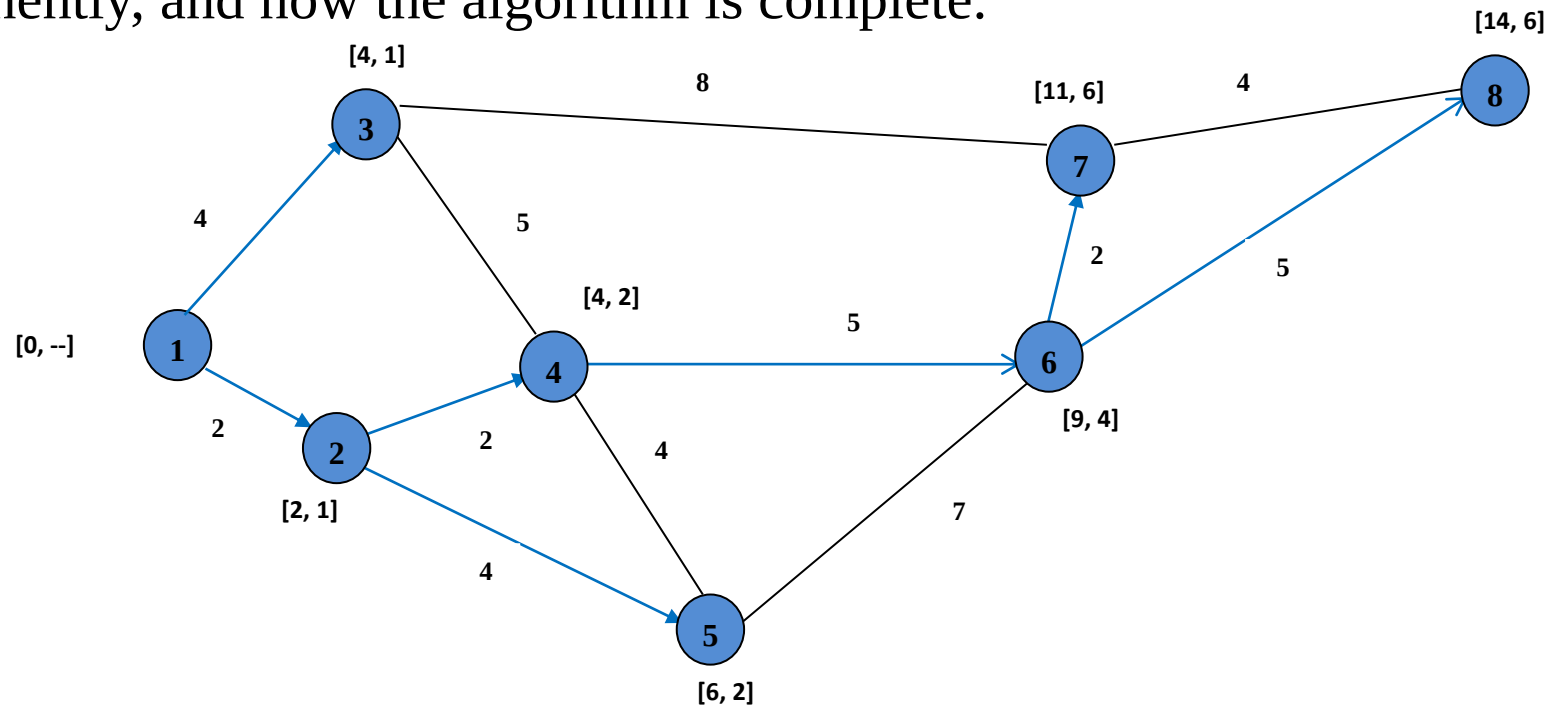

Figure 10: Final Iteration with Minimum Links with Travel Time from Node 1 to Other Nodes

Now all nodes in the list have attained their status as permanent. Thus, the maximum iteration for the problem is obtained. Therefore the solution to the shortest route from node 1 to other nodes in the network can be seen in table 3. To get the shortest distance for other nodes, we make the required node our starting node and repeat the iteration. By backtracking through the nodes using the information given by the permanent labels, the links and the associate link impedances can be obtained. After carrying out dijktra's algorithm from designated starting nodes 2, 3, 4, 5, 6, 7, 8 to all other nodes, the results are also seen in table 3.

Table 3: Showing the results of shortest route and the travel time between nodes 1, 2, 3, 4,5, 6, 7, 8 and other nodes

FROM	TO	MINIMUM LINK PATH	TRAVE TIME
1	2	1 - 2	2
	3	1 - 3	4
	4	1 - 2 - 4	4
	5	1 - 2 - 5	6
	6	1 - 2 - 4 - 6	9
	7	1 - 2 - 4 - 6 - 7	11
	8	1 - 2 - 4 - 6 - 8	14
2	1	2 - 1	8
	3	2 - 1- 3	6
	4	2 - 4	2
	5	2 - 5	4
	6	2 - 4 - 6	7
	7	2 - 4 - 6 - 7	9
	8	2 - 4 - 6 - 8	12

FROM	TO	MINIMUM LINK PATH	TRAVE TIME
3	1	3 - 1	4
	2	3 - 1- 2	6
	4	3 - 4	5
	5	3 - 4 - 5	9
	6	3 - 4 - 6	10
	7	3 - 7	8
	8	3 - 7 - 8	12
4	1	4 - 2 - 1	4
	2	4 - 2	2
	3	4 - 3	5
	5	4 - 5	4
	6	4 - 6	5
	7	4 - 6 - 7	7
	8	4 - 6 - 8	10

FROM	TO	MINIMUM LINK PATH	TRAVE TIME
5	1	5 - 2 - 1	6
	2	5 - 2	4
	3	5 - 4 - 3	9
	4	5 - 4	4
	6	5 - 6	7
	7	5 - 6 - 7	9
	8	5 - 6 - 8	12
6	1	6 - 4 -2 - 1	9
	2	6 - 4 - 2	7
	3	6 - 4 -3	10
	4	6 - 4	5
	5	6 - 5	7
	7	6 - 7	2
	8	6 - 8	5

FROM	TO	MINIMUM LINK PATH	TRAVE TIME
7	1	7- 6 - 4 - 2 -1	11
	2	7- 6 - 4 - 2	9
	3	7 - 3	8
	4	7 - 6 - 4	7
	5	7 - 6 - 5	9
	6	7 - 6	2
	8	7 - 8	4
8	1	8 - 6- 4 - 2 -1	14
	2	8 - 6 - 4 - 2	12
	3	8 - 7 - 3	12
	4	8 - 6 - 4	10
	5	8 - 6 - 5	12
	6	8 - 6	5
	7	8 - 7	4

6.2 Floyd's Algorithm

Table 4: Showing Matrix T_0 and Matrix S_0 (times and paths of the network)

T_0 =

	1	2	3	4	5	6	7	8
1	-	2	4	∞	∞	∞	∞	∞
2	2	-	∞	2	4	∞	∞	∞
3	4	∞	-	5	∞	∞	8	∞
4	∞	2	5	-	4	5	∞	∞
5	∞	4	∞	4	-	7	∞	∞
6	∞	∞	∞	5	7	-	2	5
7	∞	∞	8	∞	∞	2	-	4
8	∞	∞	∞	∞	∞	5	4	-

S_0 =

	1	2	3	4	5	6	7	8
1	-	1	1	1	1	1	1	1
2	2	-	2	2	2	2	2	2
3	3	3	-	3	3	3	3	3
4	4	4	4	-	4	4	4	4
5	5	5	5	5	-	5	5	5
6	6	6	6	6	6	-	6	6
7	7	7	7	7	7	7	-	7
8	8	8	8	8	8	8	8	-

All elements along the main diagonal of matrix T_0 equal zero since by definition $t^0_{12}=0$ for i = j. We note element t^0_{12} and t^0_{13} of matrix T_0 have elements equal to 2 and 4 since the length of the branches connecting nodes 1 – 2 and 1-3 are 2 and 4 respectively. Element t^0_{14} equals infinity since the network has no branch which is oriented from node 1 to node 4. Element t^0_{15} of matrix T_0 equals infinity as well since there is direct branch linking nodes 1 and 5 and so on.

We also note that node i is the immediate predecessor of node j on the shortest path leading from node i to node j (for $i \neq j$). For this reason we have, elements of $s^0_{12} = s^0_{13} = s^0_{14} \ldots = 1$ and $s^0_{21} = s^0_{23} = s^0_{24} \ldots = 2$ and so on in matrix S_0

The iterations of Floyd's algorithm look as shown below:

Table 5: Matrices T_1 and S_1 (first iteration)

T_1 =

	1	2	3	4	5	6	7	8
1	-	2	4	∞	∞	∞	∞	∞
2	2	-	6	2	4	∞	∞	∞
3	4	6	-	5	∞	∞	8	∞
4	∞	2	5	-	4	5	∞	∞
5	∞	4	∞	4	-	7	∞	∞
6	∞	∞	∞	5	7	-	2	5
7	∞	∞	8	∞	∞	2	-	4
8	∞	∞	∞	∞	∞	5	4	-

S_1 =

	1	2	3	4	5	6	7	8
1	-	1	1	1	1	1	1	1
2	2	-	1	2	2	2	2	2
3	3	1	-	3	3	3	3	3
4	4	4	4	-	4	4	4	4
5	5	5	5	5	-	5	5	5
6	6	6	6	6	6	-	6	6
7	7	7	7	7	7	7	-	7
8	8	8	8	8	8	8	8	-

Table 6: Matrices T_2 and S_2 (second iteration)

T_2 =

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	∞	∞	∞
2	2	-	6	2	4	∞	∞	∞
3	4	6	-	5	10	∞	8	∞
4	4	2	5	-	4	5	∞	∞
5	6	4	10	4	-	7	∞	∞
6	∞	∞	∞	5	7	-	2	5
7	∞	∞	8	∞	∞	2	-	4
8	∞	∞	∞	∞	∞	5	4	-

S_2 =

	1	2	3	4	5	6	7	8
1	-	1	1	2	2	1	1	1
2	2	-	1	2	2	2	2	2
3	3	1	-	3	2	3	3	3
4	2	4	4	-	4	4	4	4
5	2	5	2	5	-	5	5	5
6	6	6	6	6	6	-	6	6
7	7	7	7	7	7	7	-	7
8	8	8	8	8	8	8	8	-

Table 7: Matrices T_3 and S_3 (third iteration)

T_3=

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	∞	12	∞
2	2	-	6	2	4	∞	14	∞
3	4	6	-	5	10	∞	8	∞
4	4	2	5	-	4	5	13	∞
5	6	4	10	4	-	7	18	∞
6	∞	∞	∞	5	7	-	2	5
7	12	14	8	13	18	2	-	4
8	∞	∞	∞	∞	∞	5	4	-

S_3=

	1	2	3	4	5	6	7	8
1	-	1	1	2	2	1	3	1
2	2	-	1	2	2	2	3	2
3	3	1	-	3	2	3	3	3
4	2	4	4	-	4	4	3	4
5	2	5	2	5	-	5	3	5
6	6	6	6	6	6	-	6	6
7	3	3	7	3	3	7	-	7
8	8	8	8	8	8	8	8	-

Table 8: Matrices T_4 and S_4 (fourth iteration)

T_4=

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	9	12	∞
2	2	-	6	2	4	7	14	∞
3	4	6	-	5	9	10	8	∞
4	4	2	5	-	4	5	13	∞
5	6	4	9	4	-	7	17	∞
6	9	7	10	5	7	-	2	5
7	12	14	8	13	17	2	-	4
8	∞	∞	∞	∞	∞	5	4	-

S_4=

	1	2	3	4	5	6	7	8
1	-	1	1	2	2	4	3	1
2	2	-	1	2	5	4	3	2
3	3	1	-	3	4	4	3	3
4	2	2	3	-	4	4	3	4
5	2	5	4	5	-	5	4	5
6	4	4	4	6	6	-	6	6
7	3	3	7	3	4	7	-	7
8	8	8	8	8	8	8	8	-

Table 9: Matrices T_5 and S_5 (fifth iteration)

T_5=

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	9	12	∞
2	2	-	6	2	4	7	10	∞
3	4	6	-	5	9	10	8	∞
4	4	2	5	-	4	5	13	∞
5	6	4	9	4	-	7	17	∞
6	9	7	10	5	7	-	2	5
7	12	10	8	13	17	2	-	4
8	∞	∞	∞	∞	∞	5	4	-

S_5=

	1	2	3	4	5	6	7	8
1	-	1	1	2	2	4	3	1
2	2	-	1	2	2	4	3	2
3	3	1	-	3	4	4	3	3
4	2	4	4	-	4	4	3	4
5	2	5	4	5	-	5	4	5
6	4	4	4	6	6	-	6	6
7	3	3	7	3	4	7	-	7
8	8	8	8	8	8	8	8	-

Table 10: Matrices T_6 and S_6 (sixth iteration)

T_6=

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	9	11	14
2	2	-	6	2	4	7	9	12
3	4	6	-	5	9	10	8	15
4	4	2	5	-	4	5	7	10
5	6	4	9	4	-	7	9	12
6	9	7	10	5	7	-	2	5
7	11	9	8	7	9	2	-	4
8	14	12	15	10	12	5	4	-

S_6=

	1	2	3	4	5	6	7	8
1	-	2	3	2	2	4	6	6
2	1	-	1	2	2	4	6	6
3	3	1	-	3	4	4	3	6
4	2	4	4	-	4	4	6	6
5	2	5	4	5	-	5	6	6
6	4	4	4	6	6	-	6	6
7	6	6	7	6	6	7	-	8
8	6	6	6	6	6	6	8	-

Table 11: Matrices T_7 and S_7 (third iteration)

T_7=

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	9	11	14
2	2	-	6	2	4	7	9	12
3	4	6	-	5	9	10	8	12
4	4	2	5	-	4	5	7	10
5	6	4	9	4	-	7	9	12
6	9	7	10	5	7	-	2	5
7	11	9	8	7	9	2	-	4
8	14	12	12	10	12	5	4	-

S_7=

	1	2	3	4	5	6	7	8
1	-	2	3	2	2	4	6	6
2	1	-	1	2	2	4	6	6
3	3	1	-	3	4	4	3	7
4	2	4	4	-	4	4	6	6
5	2	5	4	5	-	5	6	6
6	4	4	4	6	6	-	6	6
7	6	6	7	6	6	7	-	7
8	6	6	7	6	6	8	8	-

Table 12: Matrices T_8 and S_8 (eighth iteration)

T_8=

	1	2	3	4	5	6	7	8
1	-	2	4	4	6	9	11	14
2	2	-	6	2	4	7	9	12
3	4	6	-	5	9	10	8	12
4	4	2	5	-	4	5	7	10
5	6	4	9	4	-	7	9	12
6	9	7	10	5	7	-	2	5
7	11	9	8	7	9	2	-	4
8	14	12	12	10	12	5	4	-

S_8=

	1	2	3	4	5	6	7	8
1	-	1	1	2	2	4	6	6
2	2	-	1	4	5	4	6	6
3	3	1	-	3	4	4	3	7
4	2	4	4	-	4	4	6	6
5	2	5	4	5	-	5	6	6
6	4	4	4	6	6	-	6	6
7	6	6	3	6	6	7	-	7
8	6	6	7	6	6	8	8	-

7.0 Discussion of Result

Dijkstra's algorithm is designed to determine the shortest routes between the source node and every other node in the network. Floyd's algorithm is more general than Dijkstra's because it determines the shortest route between any two nodes in the network. The result of the study has shown that table 12 holds the same solution as table 3

8.0 Recommendation

Based on the findings and results of this study, we recommend that, commercial motorcycles should follow the links shown in Table 3 to enable them save time and fuel in their business of town service in Bida town. Management of other transport agencies like Tricycles in other towns should consult us for proper implementation of this study in their towns to save time and fuel for commercial transporters who are engaged in town service.

9.0 Conclusion

Although the case study of this research is strictly on Motorcycles, it can be concluded that the study could be generalized to any other means of transportation in Bida. The study provided the present and future motorcycles and other road users with minimum link paths in the town. It can be clearly seen from the study that Esso and Lunchita happens to be the shortest route to most of the routes in the town. Finally, it is discovered that both algorithms produce the same solution, although Dijkstra's algorithm is designed to determine the shortest routes between the source node and every other node in the network while Floyd's algorithm determines the shortest route between any two nodes in the network.

References

Bellman, R. (1958).On a Routing Problem. *Quarterly of Applied Mathematics* **16(1),** 87–90. (http:/ / www.ams. org/ mathscinet-getitem?mr=0102435).

Chen, K. M. (2009). A Real-Time Wireless Route Guidance System for Urban Traffic Management and its Performance Evaluation.*70th Vehicular Technology Conference Anchorage*, Pp 1-5.

Floyd, R. W. (1962). Algorithm 97: Shortest Path. *Communications of the ACM* **5** (6): 345. doi: 10.1145/367766.368168 (http:/ /dx. doi. org/ 10. 1145/ 367766. 368168).

Meghanathan, D. N. (n.d.).Review of Graph Theory Algorithms.*MS: Department of Computer Science ,Jackson State University*.

Moore, Edward F. (1959). The Shortest Path Through a Maze. *Proc. Internat. Sympos. Switching Theory Part II*. Cambridge, Mass.: Harvard Univ. Press. pp. 285–292. MR 0114710 (http:/ / www. ams. org/mathscinet-getitem?mr=0114710).

References

Roy, B. (1959). “Transitivity et connexite”. *C. R. Acad. Sci. Paris* **249**: 216–218.

Sadeghi-Niaraki, A., Varshosaz, M., Kim, K., and Jung, J. (2011). Real World Representation of a Road Network for Route Planning in GIS. *Expert Systems with Applications*, 38 (10), 11999-12008.

Warshall, S. (1962). “A Theorem on Boolean Matrices”. *Journal of the ACM* **9**(1), 11–12. doi: 10.1145/321105.321107 (http:/ / dx. doi. org/ 10. 1145/ 321105. 321107).

THANK YOU FOR LISTINING