Applied Mathematics and Computational Intelligence
Volume 14, No. 2, 2025 [145 – 158]

Modeling Road Network in the Main Campus of Universiti Putra Malaysia, Serdang, Selangor, Using Graph Theory

Tan Chai Fang[1], Athirah Nawawi[1*], Siti Hasana Sapar[1]

[1]Department of Mathematics and Statistics, Faculty of Science, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia

* Corresponding author: athirah@upm.edu.my

Received: 19 November 2024
Revised: 6 February 2025
Accepted: 8 May 2025

ABSTRACT

Graph theory is a powerful mathematical tool that can be applied to solve many real-life problems including modeling traffic flows as directed graphs and analyzing them to propose solutions for congestion problems. This research aims to describe the road network of the north and south campuses of Universiti Putra Malaysia, Serdang, Selangor, as a directed graph, consisting of junctions as vertices and interchanges (or links) between junctions with other junctions as directed edges. This research also aims to determine the shortest path between a junction to all other junctions by using two shortest path algorithms, namely the Dijkstra and Floyd-Warshall algorithms, and consequently compare their efficiencies in producing the results. The algorithms are modified so that not only the length of the shortest path is given but also to identify the shortest path itself. Based on the findings of this research, we propose several strategies to minimize traffic congestion, especially during peak hours or convocation sessions, which would benefit both university and surrounding communities.

Keywords: Dijkstra Algorithm, Floyd-Warshall Algorithm, Shortest Path.

1 INTRODUCTION

The application of graph theory to address traffic congestion at the main campus of Universiti Putra Malaysia (UPM) involves modeling the road network as a directed multigraph, reflecting the complex interactions between junctions and road segments. This approach allows for a comprehensive analysis of traffic flow dynamics, considering factors such as one-way roads and multiple roadways connecting different junctions. By assigning weights to the edges in the directed multigraph, representing distances or costs associated with traversing each road segment, the model can accurately capture the intricacies of travel within the main campus of UPM. These weights facilitate the identification of the shortest or most efficient paths between junctions, allowing the development of strategies to minimize congestion and optimize traffic flow.

The degree of each vertex in the graph model serves as a crucial indicator of traffic flow at junctions, highlighting areas prone to congestion. By analyzing vertex degrees, it becomes possible to pinpoint junctions experiencing high traffic volumes, allowing for targeted interventions such as installing

traffic lights or designing diversion routes. Furthermore, the distinction between trails and paths in the context of the road network delineates between routes where vertices or edges may be repeated, informing decisions to ensure drivers follow optimal, nonrepetitive paths for efficient navigation.

Directed multigraphs accurately represent the directionality of traffic flow in each segment of the road, providing a nuanced understanding of travel patterns within the main campus of UPM. This detailed representation facilitates the development of traffic management plans tailored to specific routes and junctions, contributing to improved overall efficiency and reduced travel times for the university community. By leveraging graph theory principles and methodologies, UPM can devise effective solutions to alleviate congestion, enhancing productivity and convenience for students, faculty, and staff navigating the campus road network.

Hence, in Section 2 of this article, we discuss on the research methodology which includes modelling of UPM road network as a directed graph and applying both Dijkstra's and Floyd-Warshall Algorithms on the resulting graph. Subsequently, in Section 3, we present the description of our results and discuss the analysis based on the findings. Finally, the Conclusion section highlights the successful modelling of the UPM road network and demonstrates the effectiveness of the Floyd-Warshall algorithm in efficiently determining the shortest paths.

1.1 Related Research

[1] proposed a methodology for improving city traffic management using directed graphs, focusing on two main strategies. Firstly, they aimed to enhance road efficiency by reducing traffic congestion at bottleneck areas, achieved by minimizing interruptions at junctions with multiple incoming flows. This approach resulted in a redesigned city map with smoother traffic flows and decreased congestion. Secondly, traffic accumulation at junctions was mitigated by regulating traffic flow through the timing patterns of traffic lights, based on lane capacities, leading to a smoother driving experience and optimal traffic redistribution. Additionally, [1] explored the use of graph theory, particularly edge and vertex connectivity, to address traffic control issues at junctions, proposing the placement of traffic sensors to minimize driver waiting times, especially at junctions with multiple traffic streams.

On a related note, [2] discussed the shortest path problem, which involves identifying the shortest route between an initial vertex and a final destination in a graph. Graphs represent this problem with vertices and edges, where weights assigned to edges help calculate the shortest path between vertices. This path connects the specified pair of vertices and represents the shortest route from an initial vertex to a destination. In this article as well, Dijkstra's algorithm emerges as a valuable tool for determining the shortest distance in directed graphs with non-negative edge weights. This greedy algorithm efficiently calculates the smallest possible weights of each vertex, aiding in finding the shortest path and reducing traffic congestion in certain road segments by identifying alternative routes with smaller weights. The effectiveness of Dijkstra's algorithm is further explored in the development of real-time information and navigation systems accessible via mobile phones, offering alternative routes in case of traffic congestion, accidents, or natural disasters. Additionally, studies have utilized Dijkstra's algorithm to identify the nearest distance from a user's location to specific destinations, such as specialist doctors' practices, aiming to minimize search time and ensure timely access to treatment. These applications of Dijkstra's algorithm highlight its

effectiveness in facilitating route optimization and resolving traffic congestion issues. Moreover, research demonstrates its ability to generate alternate routes for drivers to avoid congestion on specific roads, as observed in the study conducted in Purwokerto.

In graph theory, the Floyd-Warshall algorithm, as outlined by [3], stands as a prominent method for determining the shortest routes between any pair of vertices within a weighted graph. It operates by creating a distance matrix, utilizing dynamic programming to iteratively update it until all potential pathways are considered. The algorithm initializes the distance matrix with direct edge weights between vertices, then proceeds to determine shorter paths by calculating intermediate vertices for each vertex pair. Despite its ability to handle networks with positive or negative edge weights, it cannot manage graphs with negative cycles and is best suited for small to medium-sized networks. Nevertheless, the Floyd-Warshall algorithm finds extensive application in various fields such as network routing, traffic optimization, and graph connectivity analysis due to its capability in determining shortest pathways between any vertex pairs.

Additionally, [4] highlight the algorithm's efficiency in calculating minimum path lengths for all pairs of vertices in a directed graph, contrasting it with Dijkstra's algorithm, which may be slower due to blind search but is effective in solving the shortest path problem.

Furthermore, [5] stated that the Floyd-Warshall algorithm is a highly effective solution for solving the shortest path problem. Unlike Dijkstra's algorithm, which can be slow due to its blind search approach, the Floyd-Warshall algorithm efficiently handles route searches. Hence, in this research, both Dijkstra's and Floyd-Warshall algorithms will be used to determine the shortest paths between vertices in the graph of UPM main campus, and the efficiency of the algorithms will be compared by measuring the computation time. Both algorithms are intended to find the shortest path lengths but do not provide the actual paths to be taken. Therefore, in this research, Dijkstra's and Floyd-Warshall algorithms will be modified to demonstrate the shortest path from one vertex to another, and the results from both algorithms will be compared to assess their accuracy. In conclusion, Dijkstra's and the Floyd-Warshall algorithms are distinct methods for solving the shortest path problem in weighted graphs. Dijkstra's algorithm identifies the shortest path from a single source vertex to all other vertices using a greedy approach that iteratively updates distance estimates. In contrast, the Floyd-Warshall algorithm calculates the shortest paths between all pairs of vertices simultaneously, employing an iterative matrix-based method that refines distance estimates by considering intermediate vertices. The key difference is that while Dijkstra's algorithm handles one shortest path at a time, Floyd-Warshall provides all shortest path distances in a single process.

2 MATERIAL AND METHODS

2.1 Modelling a Road Network into a Directed Graph

Parameter needed

$G = (V, E)$ is a directed graph to model UPM South and North maps, where V is a set of vertices in G representing the junctions of the road network in UPM Map and E is a set of edges in G

representing the direction of the traffic flow at each junction. In addition, $w(e)$ is the weight of an edge e in E of G representing the distance of the road segment.

Step to Model the Roadway as a Directed Graph

1. We collect information about the roadway such as the number of junctions, the location of each junction, the direction of traffic flow at each junction, and the length of each road segment between junctions.

2. We assign a vertex for each junction in the map, and every vertex will be labelled as v_i, which $i = 0, 1, 2, \ldots, n-1$, n is the total number of vertex in the graph.

3. Directed edges are drawn between the vertices based on the direction of traffic flow at each junction. If traffic flows in both directions, a parallel edge with different directions will be created between the vertices to represent the traffic flow in each direction.

4. We assign weights to the edges based on the length between the junctions. Distance is measured by using satellite Google Maps. This will be used to calculate the shortest path between two vertices in the graph.

2.2 Dijkstra's Algorithm

Let G be a simple and connected weighted graph. The weights are non-negative real numbers and are denoted by W. Let a and z be two vertices in G. Let P be a path in G from a to z. The length of path P, denoted as $L(P)$, is the sum of the weights of all edges on path P. Our objective in this section is to determine the length of the shortest path from a to z. Dijkstra's algorithm iteratively constructs the set S that consists of all vertices of G for which the length of the shortest path has been determined. The algorithm ([6]) is described in the following steps:

1. $S := \emptyset$.

2. With V being the set of all vertices of G, let $N := V$ in the beginning. (Generally, $V = S \cup N$)

3. Since a is the origin vertex, $L(a)$ which is the length of a shortest path from a to a is 0, i.e, $L(a) = 0$.

4. For all vertices $u \in V$, $u \neq a$ and $L(u) := \infty$.

5. While the end vertex $z \notin S$, iteratively do

   (a) Let $v \in N$ such that $L(v) = \min \{L(u) | u \in N\}$.

   (b) $S := S \cup \{v\}$.

   (c) $N := N - \{v\}$.

   (d) For all $w \in N$ such that there is an edge from v to w. If $L(v) + W[u, w] < L(w)$ then $L(w) = L(v) + W[u, w]$.

6. Algorithm will be terminated once the end vertex z is in S and $L(z)$ gives the length of a shortest path from a to z.

2.3 Floyd-Warshall Algorithm

Let G be a directed and weighted graph (V, E), consisting of a set of vertices, V, and edges, E. Each edge e, is assigned a weight denoted as $w(e)$. The weight matrix of the graph, W_{ij}, must not contain a negative weight cycle. This algorithm computes the smallest weight among all paths connecting pairs of vertices simultaneously, iteratively assessing each pair until the optimal route to reach each destination vertex is determined with minimum weights. The algorithm ([7]) is described in the following steps:

1. Let A be the weight matrix of dimension $n \times n$ representing the distances between vertices, with n being the number of vertices in the graph G.

2. The entry of the matrix A at the position of i and j ranging from 1 to n, is denoted by $A_{[i][j]}$. With k being the intermediate vertex between i and j ranging from 1 to n, $A_{[i][j]}$ is updated as follows:

   $$A_{[i][j]} = \min(A_{[i][j]}, A_{[i][k]} + A_{[k][j]}).$$

3. Terminate when all ∞ and all other entries within the matrix have been replaced by the shortest distances.

4. The final matrix A contains the shortest path distances between all pairs of vertices.

3 RESULTS AND DISCUSSION

3.1 Modelling the UPM Network as a Directed Graph

Junctions are identified using the UPM map – covering both North and South Campuses. Each junction represents a vertex, labelled as vertices v_i, where $i = 0, 1, 2, \ldots, 51$, as shown in Figure 1. These vertices are connected by directed edges that represent the roadways between junctions, with edge directions determined by traffic flow. The inclusion of edges to the graph is limited to roadways accessible by car, identified through satellite in Google Maps, excluding paths restricted to pedestrians or cyclists.

As a result, Figure 2 shows the directed graph representing the UPM traffic network, denoted as G, comprises of set $V(G)$ and $E(G)$. Its set $V(G)$ contains a total of 52 vertices. Meanwhile, $E(G)$ is the edge set of G, consists of a total of 109 directed edges within the graph. This directed graph is also a multigraph which may contain parallel edges between vertices. The edges of the graph G are also labelled with their weights. The weights of these edges are the length of each road segment which are measured using Google Maps satellite imagery, providing distance measurements in meters (m). Additionally, these distances become the entries of the weight matrix, of dimension 52×52, which later being utilized to compute shortest path distances. In this weight matrix, non-adjacent vertices are designated with a weight of infinity, denoted as INF.

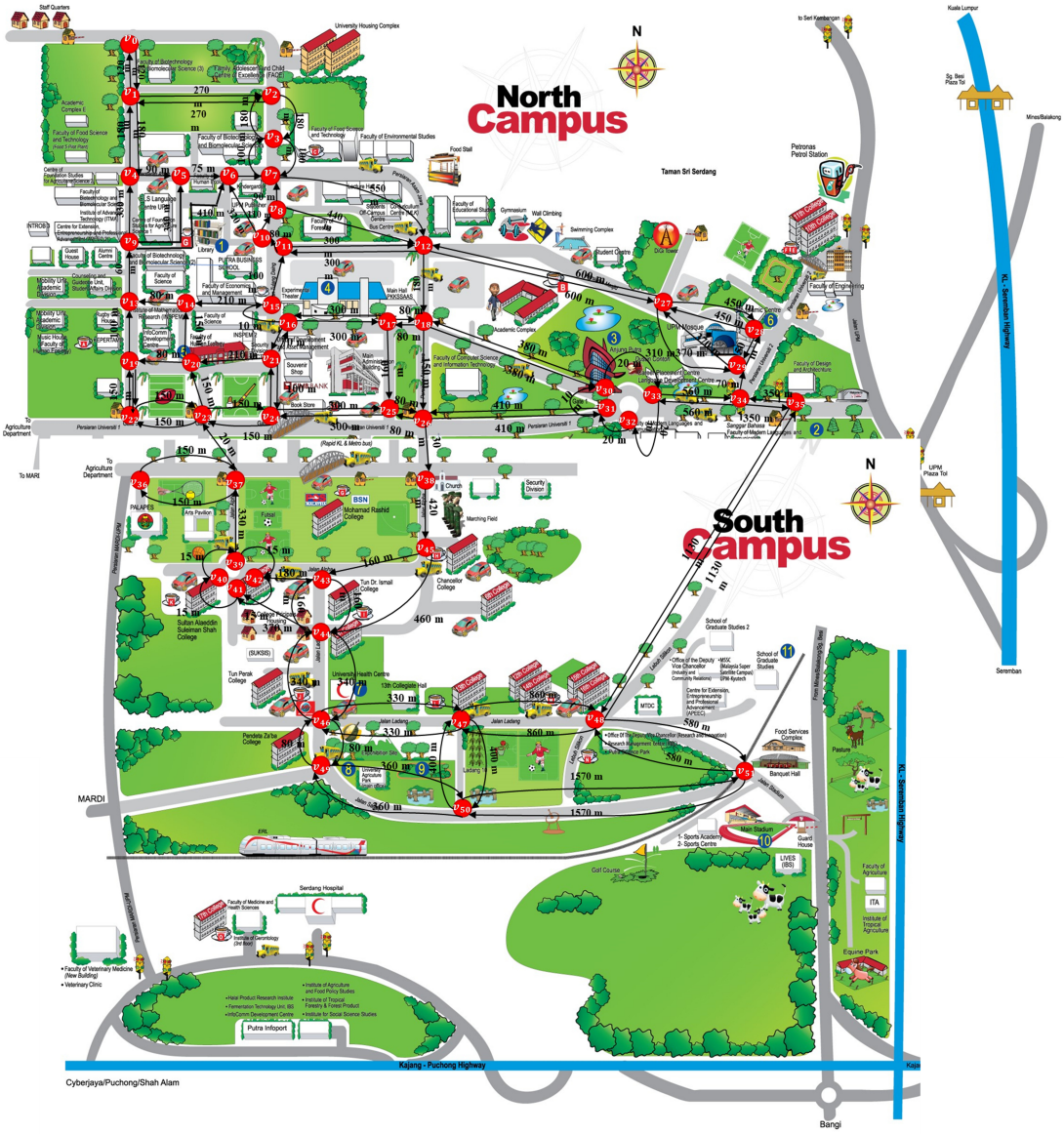

Figure 1 : UPM Map-Combining North and South Campuses

3.2 C++ Coding of Dijkstra Algorithm to Find the Shortest Path of UPM Map

The modified Dijkstra's algorithm incorporates enhancements to facilitate the printing of the shortest path and the computation of total execution time. By utilizing *std* :: *chrono* :: high resolution clock, time points are captured before and after the execution of the algorithm segment. The duration of code execution is then calculated by subtracting the start time from the end time and converting the result to microseconds. This duration serves as a metric for assessing the computational efficiency of the algorithm and enables quantitative comparisons between different implementations. Additionally, the algorithm includes functions such as printSolution and printPath,

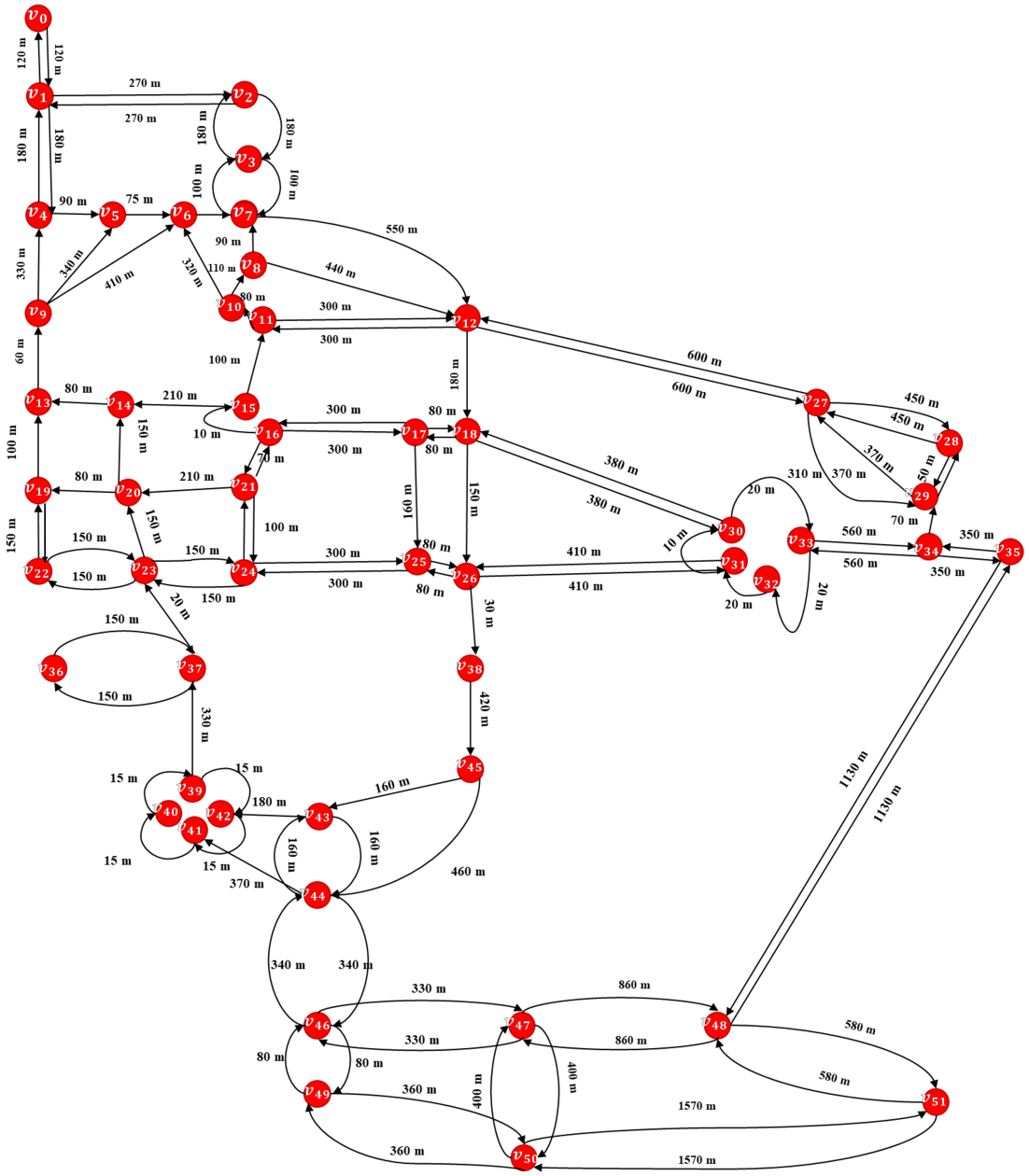

Figure 2 : Directed Graph of UPM Map

which work in tandem to display the shortest distances and paths from the source vertex to all other vertices. The printPath function utilizes recursion to trace back the shortest path by following parent vertices from the destination to the source. These enhancements contribute to a more comprehensive evaluation of the algorithm's effectiveness, enabling clear comparisons of routes based on both their lengths and computation times.

```
#include <chrono>
int minDistance(int dist[], bool sptSet[]) {
    int min = INT_MAX, min_index;
```

```
    for (int v = 0; v < V; v++)
        if (!sptSet[v] && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}
void printPath(int parent[], int j) {
    if (parent[j] == -1)
        return;
    printPath(parent, parent[j]);
    printf(" -> %d", j);
}
void printSolution(int dist[], int parent[], int src) {
    printf("Vertex   Distance from Source (m)                    Path\n");
    for (int i = 0; i < V; i++) {
        if (i != src) {
            printf("%d \t\t %d \t\t\t\t%d", i, dist[i], src);
            printPath(parent, i);
            printf("\n");
        }
    }
}
void dijkstra(int graph[V][V], int src, long long& total_time) {
    int dist[V];
    bool sptSet[V];
    int parent[V];
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = false;
        parent[i] = -1;
    }
    dist[src] = 0;
    auto start_time = std::chrono::high_resolution_clock::now();
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++) {
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]
            + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u;
            }
        }
    }
    auto end_time = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::
    microseconds>(end_time - start_time);
    printf("Computation Time from vertex %d: %lld microseconds\n",
    src, static_cast<long long>(duration.count()));
    total_time += duration.count();
    printSolution(dist, parent, src);
}
int main() {
    int graph[V][V] = { .....Insert the Weight Matrix.....};
    long long total_time = 0;
    for (int i = 0; i < V; i++) {
        dijkstra(graph, i, total_time);
```

```
    }
    printf("Total Computation Time for all vertices: %lld microseconds\n",
    total_time);
    return 0;
}
```

3.3 C++ Coding of Floyd-Warshall Algorithm to Find the Shortest Path of UPM Map

The modified Floyd-Warshall algorithm incorporates enhancements to display both the shortest paths and their corresponding computation times. Computation time is measured using $< chrono >$ in C++ to accurately assess the algorithm's performance. The program captures time points before and after executing the algorithm segment using $std :: chrono ::$ high resolution clock, calculating the duration of code execution in microseconds. This duration serves as a metric for evaluating the computational efficiency of the algorithm. Additionally, the algorithm includes functions such as printMatrix and printAllShortestPaths to display the computed shortest path matrix and showcase all shortest paths between pairs of vertices, respectively. These modifications enable users to identify optimal routes with minimum length while providing insights into the algorithm's effectiveness based on computation times.

```
#include <chrono>
void printMatrix(int matrix[][nV]);
void printAllShortestPaths(int parent[][nV]);

 void floydWarshall(int graph[][nV]) {
    int matrix[nV][nV], parent[nV][nV];
    int i, j, k;
    for (i = 0; i < nV; i++) {
        for (j = 0; j < nV; j++) {
            matrix[i][j] = graph[i][j];
            parent[i][j] = i; // Initialize the parent array
        }
    }
    auto start_time = std::chrono::high_resolution_clock::now();
    for (k = 0; k < nV; k++) {
        for (i = 0; i < nV; i++) {
            for (j = 0; j < nV; j++) {
                if (matrix[i][k] + matrix[k][j] < matrix[i][j]) {
                    matrix[i][j] = matrix[i][k] + matrix[k][j];
                    parent[i][j] = parent[k][j];
                    // Update the parent for shortest path
                }
            }
        }
    }
    auto end_time = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>
    (end_time - start_time);
    printf("Computation Time: %lld microseconds\n", static_cast<long long>
    (duration.count()));
    printMatrix(matrix);
    printf("Shortest Paths for All Vertices:\n\n");
    printAllShortestPaths(parent);
}
```

```
void printMatrix(int matrix[][nV]) {
    printf("Shortest Path Matrix:\n");
    for (int i = 0; i < nV; i++) {
        for (int j = 0; j < nV; j++) {
            if (matrix[i][j] == INF)
                printf("INF\t");
            else
                printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}
void printAllShortestPaths(int parent[][nV]) {
    for (int source = 0; source < nV; source++) {
        for (int destination = 0; destination < nV; destination++) {
            if (source != destination) {
                printf("Shortest Path from %d to %d: ", source, destination);
                std::vector<int> path;
                int current = destination;
                while (current != source) {
                    path.push_back(current);
                    current = parent[source][current];
                }
                path.push_back(source);
                for (int i = path.size() - 1; i >= 0; i--) {
                    printf("%d", path[i]);
                    if (i > 0) {
                        printf(" -> ");
                    }
                }
                printf("\n");
            }
        }
    }
}
int main() {
    int graph[nV][nV] = {  .....Insert the Weight Matrix.....  };

    floydWarshall(graph);

    return 0;
}
```

3.4 Shortest Path between Vertices in the Graph and Their Distances

The computational results obtained through the implementation of the Dijkstra and Floyd-Warshall Algorithms in C++ provide significant insights into the shortest path computations within the UPM roadway network. Although both algorithms yield identical outputs in terms of the calculated shortest paths, they distinctly showcase variations in computational times. In Table 1, the source vertex (the initial vertex), the destination (the end vertex), and the corresponding shortest path distances are measured in meters. Interpreting the shortest path is straightforward; for instance, $v_0 - v_1 - v_2$ denotes the path originating from v_0, passing through v_1, and then reaching the destination v_2.

Table 1 : Shortest paths between v_0 to v_i $(1 \leq i \leq 5)$ and its distance

Source Vertex	**Destination Vertex**	**Distance** (m)	**Shortest Path**
v_0	v_1	120	$v_0 - v_1$
	v_2	390	$v_0 - v_1 - v_2$
	v_3	570	$v_0 - v_1 - v_2 - v_3$
	v_4	300	$v_0 - v_1 - v_4$
	v_5	390	$v_0 - v_1 - v_4 - v_5$

The rest of the information on the shortest paths involving all vertices in the directed graph of UPM map can be accessed through the QR code in Figure 3.

Figure 3 : Shortest path between a vertex to another vertex in the graph G

Both Dijkstra's and Floyd-Warshall algorithms consistently identify the same shortest path within the UPM traffic network. While both algorithms yield identical outputs for the shortest path and weight, they exhibit notable differences in computational efficiency. Dijkstra's Algorithm takes approximately 2384 microseconds to compute all paths, while the Floyd-Warshall algorithm completes this task in just 600 microseconds due to its matrix-based approach.

Further analysis of the data reveals that the longest shortest path between two vertices in UPM spans from v_{51} to v_{15}, covering a distance of 4610 meters, which is the diameter of the traffic network. Additionally, the shortest path covering the most vertices extends from v_{50} to v_8, measuring 3520 meters. Traversing a greater number of vertices may lead to increased interruption time, despite the shorter overall path length, due to more junctions and stops along the route. Mitigating traffic congestion requires considering not only the shortest path but also the total travel time, incorporating interruption time. Further insights into minimizing waiting times are provided by [8], offering valuable strategies for traffic congestion management.

3.5 Analysis

Table 2 displays some vertices of the graph G which have high degree for relevant junctions that were identified when doing this research.

Table 2 : Vertices with high degree

Vertex	**Vertex Degree**	**Name of Junction**
v_{12}	7	Cross junction between Faculty of Educational Studies and Co-curriculum Centre (MLK)
v_1	6	Junction between Lorong Sapucaya, Persiaran Asam Jawa and Faculty of Biotechnology and Biomolecular Sciences (3)
v_{18}	6	Cross junction between Main Hall (PKKSSAAS), Academic Complex and Faculty of Computer Science and Information Technology
v_{23}	6	Gate 4
v_{24}	6	Gate 3
v_{26}	6	Gate 2
v_{27}	6	Junction on Persiaran Masjid
v_{44}	6	Junction in front of College Principals Housing
v_{46}	6	Cross junction between Pendeta Za'ba College, University Health Care and Tun Perak College
v_{47}	6	Junction between 13th College and Ladang 10
v_{48}	6	Junction between 16th College, MTDC and Office of the Deputy Vice-Chancellor (Research and Innovation)
v_{50}	6	Junction between Jalan Satelit and Ladang 10

The high degree of these vertices indicates the high possibility of traffic congestion when a substantial volume of transportation traverses these junctions concurrently. Congestion is expected during peak times, such as convocations, major events, morning rush hours, and the conclusion of office hours.

3.6 Suggestion to Minimize the Traffic Congestion

To alleviate traffic congestion at UPM, a multifaceted approach has been proposed, encompassing various strategic measures. Extending the operational hours of specific routes, such as those around junctions near high-traffic areas like Haura's Cafe and the Faculty of Science, could divert vehicles away from congested zones during peak hours. Additionally, implementing access controls based on vehicle plate numbers and promoting carpooling initiatives are suggested strategies to reduce overall traffic density at crucial junctions. Furthermore, enhancing infrastructure for alternative transportation modes like cycling and walking, collaborating with local authorities to improve road access, and optimizing traffic flow through advanced traffic signal systems represent integral components of the congestion alleviation strategy.

In conjunction with the university's transportation system enhancements, fostering behavioural changes among staff and students through incentives for alternative transportation modes and flexible work or class schedules is advocated. Furthermore, educational campaigns on traffic impact awareness, real-time traffic information dissemination, and efficient parking management systems are proposed to complement infrastructure improvements. By integrating these diverse initiatives, UPM aims to comprehensively address congestion issues, enhance traffic management practices, and cultivate a sustainable and efficient transportation ecosystem within its campus premises.

4 CONCLUSION

The research successfully achieves its objective by modeling the road network at UPM as a directed graph and conducting a comparative analysis of path-finding algorithms. Both the Floyd Warshall and Dijkstra algorithms yield accurate results for finding the shortest paths, with the former demonstrating higher efficiency in computational times. This observation aligns with existing literature, reinforcing the efficacy of the Floyd Warshall algorithm in such contexts. Additionally, the study proposes a range of alternative measures to minimize traffic congestion at junctions, considering the diverse commuting patterns of UPM's staff, lecturers, and students. While implementing vehicle limitations may pose challenges for some commuters, the strategies aim to balance congestion alleviation with sustainability goals, fostering a green learning environment and facilitating faster emergency access.

A well-structured road system not only mitigates congestion but also supports efficient emergency pathways and overall campus functioning. Leveraging the traffic network enables the establishment of emergency routes and ensures swift access to essential services like ambulances during urgent hours. Moreover, such a system streamlines transportation, promotes sustainable practices, and contributes to the optimization of the campus environment. Striking a balance between facilitating convenient access for road users and implementing congestion-reducing strategies remains crucial for enhancing efficiency and sustainability within UPM and similar environments.

ACKNOWLEDGEMENT

Thanks for all the comments and suggestions from the reviewers to improve the article.

REFERENCES

[1] L. K. R. Abanes, J. A. M. Maniago, and I. B. Jos, "Traffic management at junctions along taft avenue using graph theory," in *De La Salle University Research Congress*, 2017. [Online]. Available: https://www.dlsu.edu.ph/wp-content/uploads/pdf/conferences/research-congress-proceedings/2017/SEE/SEE-I-011.pdf

[2] R. D. Gunawan, R. Napianto, R. I. Borman, and I. Hanifah, "Implementation of dijkstra's algorithm in determining the shortest path (case study: Specialist doctor search in bandar lampung)," *International Journal of Information System and Computer Science*, vol. 3, no. 3, pp. 98–106, 2019, https://jurnal.ftikomibn.ac.id/index.php/ijiscs/article/view/768.

[3] D. Sarkar, M. Chakrabarty, A. De, and S. Goswami, "Emergency restoration based on priority of load importance using floyd–warshall shortest path algorithm," in *Computational Advancement in Communication Circuits and Systems: Proceedings of ICCACCS 2018*. Springer, 2020, pp. 59–72.

[4] I. K. L. D. Pandika, B. Irawan, and C. Setianingsih, "Application of optimization heavy traffic path with floyd-warshall algorithm," in *2018 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. IEEE, 2018, pp. 57–62.

[5] Risald, A. E. Mirino, and Suyoto, "Best routes selection using dijkstra and floyd-warshall algo-

rithm," in *2017 11th International Conference on Information & Communication Technology and System (ICTS)*. IEEE, 2017, pp. 155–158.

[6] D. Malik, M. Sen, and S. Ghosh, *Introduction to Graph Theory*. Singapore: Cengage Learning Asia Pte Ltd, 2014.

[7] I. H. Toroslu, "Improving the floyd-warshall all pairs shortest paths algorithm," 2021. [Online]. Available: https://arxiv.org/abs/2109.01872

[8] S. Tanveer, "Application of graph theory in representing and modelling traffic control problems," *International Journal of Mathematics and Computer Applications Research*, vol. 6, no. 3, pp. 29–34, 2016, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2838642.