Comparative research on all to all pairs path finding algorithms in a real-world scenario

Peter Nelson Subrata[1], *Philipus Adriel* Tandra[1], *Christopher* Owen[1], *Arvin* Yuwono[1], and *Maria Seraphina* Astriani[1*]

[1]Computer Science Department, School of Computing and Media, 11480 Bina Nusantara University, Jakarta, Indonesia

Abstract. This paper presents a comparative study of the implementation of all-to-all pairs shortest path algorithms, specifically Floyd-Warshall, Johnson's, and Dijkstra's. It contributes to a better understanding of their strengths and weaknesses in different types of applications (real-world scenarios). The research demonstrates the use of these algorithms in finding the shortest path between multiple locations using a Google Maps plotter and the Google Maps API. Older research papers have shown a comparison that shows that the Floyd-Warshall algorithm is faster than the other two algorithms in certain scenarios; however, none have brought up the real-world application of such an algorithm. The null hypothesis of this study is that the Floyd-Warshall algorithm is not suitable for use in a real-life application for finding the shortest path compared to Johnson's algorithm. The results of this study have potential applications in transportation and logistics and will provide useful insights for future work in this field.

1 Introduction

In the field of graph theory, the problem of finding the optimal route between multiple locations has been well researched, with various algorithms such as Floyd-Warshall, Johnson's algorithms, and running Dijkstra's algorithm multiple times being developed to solve this problem. However, previous research has primarily focused on comparing the efficiency of these algorithms in theoretical settings. Studies such as Abu-Ryash and Tamimi (2015) [1], Pandika et al. (2019) [2], and Amaliah et al. (2016) [3] have compared the runtime of these algorithms on different types of graphs and scenarios. However, there is a gap in research on the performance of these algorithms in practical, real-world scenarios.

This research aims to fill this gap by conducting a comparative study of the Floyd-Warshall and Johnson's algorithms in a real-world scenario using the Google Maps API and GMplot library in python. The unique approach performed is using these algorithms in a Google Maps plotter, providing a set of longitudes and latitudes that represent locations, and demonstrating how these algorithms can be used to find the shortest path to all locations at the same time in a real-life application. Additionally, a comparison will be made from the usage of the two graph representations, adjacency matrix and list, as well as modifying the algorithm to return extra values regarding the path the algorithm took to calculate the shortest path so that the main application can draw it on the map.

The null hypothesis of this test, with an alpha value of 0.05, is to determine if there is a difference in the mean runtime of the two algorithms. The research used an alpha value of 0.05 in their null hypothesis test to determine if there is a difference in the mean runtime of the two algorithms. Alpha is a threshold value used in statistics to determine the level of significance of a test. In this case, an alpha value of 0.05 indicates that the researchers are willing to accept a 5% chance of a false positive in their results. This means that if the null hypothesis is true (there is no significant difference in the mean runtime of the two algorithms), there is a 5% chance that the test will incorrectly reject the null hypothesis and find a significant difference. This is a commonly used alpha value in many statistical tests. The research will provide valuable insight into the performance of these algorithms in a practical, real-world scenario and contribute to a better understanding of their strengths and weaknesses in different types of applications.

2 Related Works

The comparison of results from using the Floyd-Warshall, Johnson and Dijkstra's algorithms have been well researched. In a 2015 study by Abu-ryash and Tamimi [1] they conducted an investigation and comparison of a variety of shortest path algorithms. The purpose was to discover the efficiency of each algorithm, highlighting the best use case for each algorithm. The Johnson's algorithm is claimed to be faster than Floyd-Warshall on sparse graph, but on the contrary, Floyd-Warshall algorithm is faster when the graph is dense. On a side note, Dijkstra's algorithm is more memory efficient for sparse graph as it doesn't need distance matrix to be represented as dense matrix.

* Corresponding author: seraphina@binus.ac.id

Pandika et al. [2] mentioned that Floyd-Warshall algorithm works effectively and efficiently in providing optimal routes affected by severe congestion, which in their scenario was in the area of Bandung where there are many tourists can get contribute to the traffic congestion in that area. As a result, they designed an Android application where they implemented Floyd-Warshall algorithm to help tourists get the optimal route to their destination and proved to have impressive runtime and had effective results.

A study conducted by Amaliah et al. [3] in 2016 demonstrated that Dijkstra algorithm is able to reach an accuracy of 92.88 using Google Maps as the reference. Both studies indicates that both algorithms work effectively. Mahmoud et al. [4] in their study compared Haversine and Vincenty formulas for a location based recommender system by stating that pathfinding algorithms like Dijkstra work well in graphs but something like the Haversine formula is more effective for getting the distance between two points on the Earth as it is neither a perfect sphere no ellipse. Afser et al. [5] implemented several path finding algorithms such as Dijkstra, Floyd-Warshall, Bellman Ford, Johnson algorithm for comparison using Genetic Algorithm.

These studies provide good insight into the research that has been done on Floyd Warshall and Johnson's algorithm. However, researchers have not extensively compared the runtimes of the Floyd-Warshall and Johnson's algorithms in real-world applications. While the algorithms have been studied and analyzed in theoretical settings, there is a lack of research on how they perform in practical scenarios. This gap in knowledge makes it difficult to determine which algorithm would be more efficient for a specific real-world problem. Further research is needed to determine the performance of these algorithms in real-world scenarios and to understand their strengths and weaknesses in different types of applications.

3 Methodology

3.1 Graph algorithm and analysis

Graph theory is a mathematical framework for modelling relationships between objects [5-6]. In the context of this research, This research is interested in finding the shortest path between different locations on a map. An adjacency matrix and an adjacency list are two common ways to represent a graph, where a graph is a set of vertices (also called nodes) and edges connecting them [7].

Graphs can be classified into 4 types: directed weighted, directed unweighted, undirected weighted, and undirected unweighted. Directed weighted graphs have edges with direction and weight, represented by arrows with tail indicating starting vertex and head indicating ending vertex. The weight represents cost or distance. The directed weighted graph is the one that the paper will mostly be using as examples as it is the one that is used most commonly in applications.

Directed weighted graphs are important in the real-life application of graph algorithms because they can be used to model a wide range of problems. For example, they can be used to model transportation networks, where the edges represent roads or flights, and the weights represent the distance or cost of traveling along that edge.

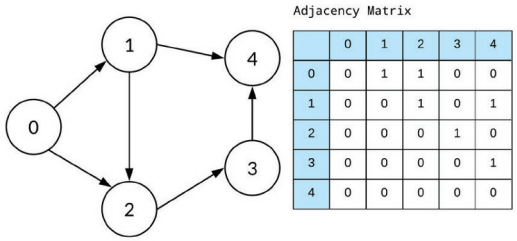

Fig. 1. An example of an adjacency matrix on a directed graph [8]

An adjacency matrix (Fig. 1) is a square matrix with the size of VxV, where V is the number of nodes in the graph. The element in the i-th row and j-th column of the matrix represents the weight of the edge between the i-th node and the j-th node. If there is no edge between nodes i and j, the corresponding element in the matrix is set to a special value such as infinity or null. Adjacency matrix representation is useful when the graph is dense, meaning that there are many edges between the nodes.

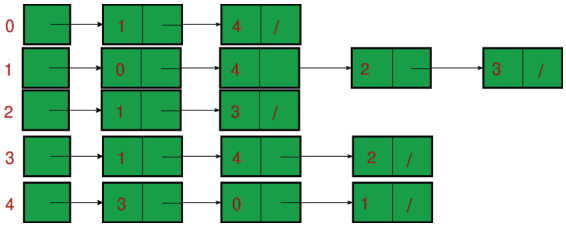

Fig. 2. An example of an adjacency list [9]

An adjacency list (Fig. 2), on the other hand, represents a graph as an array of linked lists. Each node in the graph is represented by an index in the array, and the linked list associated with that index contains the nodes that are adjacent to it. The edges are represented by the pointers in the linked lists. Adjacency list representation is useful when the graph is sparse, meaning that there are very few edges between the nodes.

Negative weight edges and cycles in graphs refer to edges or sequences of edges that have a negative cost or weight. They can be used in transportation networks, such as public transportation systems, to represent transfer points or alternative routes with lower costs [10]. In traffic engineering, they can represent alternative routes with lower travel time or fuel consumption or sequences of roads that form a loop with lower total travel time for emergency vehicles. They can also be used in traffic simulation to model and predict traffic flow by considering traffic demand and road network capacity. For example, a negative weight edge in a transportation network could represent a transfer point between two forms of transportation, where the cost of transferring is less than continuing on the same mode of transportation. In traffic engineering, a negative

weight edge could be assigned to a secondary road that has less traffic, fewer stop signs, and fewer traffic lights than the primary road, resulting in a shorter travel time. Negative weight cycles can also be used to represent a sequence of roads that form a loop and have a lower total travel time than taking the primary road. This could be useful for finding efficient routes for emergency vehicles such as ambulances or fire trucks.

Dijkstra's algorithm, a popular algorithm for finding the shortest path in a graph, is based on the principle of relaxation, where the algorithm repeatedly updates the shortest distance from the source vertex to each vertex in the graph [6]. The algorithm is not able to handle negative weight edges properly, because it can lead to the creation of negative weight cycles, which would result in an infinite loop.

Negative weight edges are problematic for Dijkstra's algorithm because they can be used to "reverse" the direction of an edge [10], making it possible for the algorithm to travel along a path that would increase the total distance, rather than decrease it. This can result in the algorithm returning an incorrect or sub-optimal solution.

In this research, a decision was made to exclude Dijkstra's algorithm and focus on other algorithms that are able to handle negative weight edges and cycles properly. This is because the paper is studying graphs that may have negative weight edges and cycles, and the research wants to ensure that the analysis and results are accurate and reliable. By using an algorithm that can handle negative weights and cycles, this research can avoid the risk of returning incorrect solutions and have a better understanding of the graph structure and behavior.

A dense graph is a graph in which the number of edges is close to the maximum possible number of edges. In other words, the ratio of the number of edges to the number of vertices is close to 1. Dense graphs have a high degree of connectivity, meaning that there are many edges between the vertices, which makes it relatively easy to navigate and traverse the graph. Below (Fig. 3) is an example of dense graph for visualization [11].

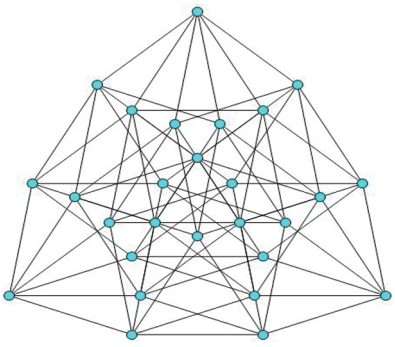

Fig. 3. An Example of a Dense Graph [12]

A sparse graph, on the other hand, is a graph in which the number of edges is relatively low compared to the maximum possible number of edges. In other words, the ratio of the number of edges to the number of vertices is relatively small. Sparse graphs have a low degree of connectivity, meaning that there are relatively few edges between the vertices, which makes it relatively difficult to navigate and traverse the graph.

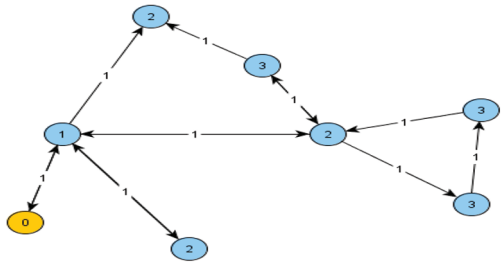

Fig. 4. An example of a sparse graph [13].

Graph density can have a significant impact on algorithm performance; dense graphs (Fig. 4) are more computationally expensive and require more memory as per [11], while sparse graphs are more efficient and require less memory. However, sparse graphs can make it more difficult to find specific paths or minimum spanning tree. Different algorithms, like Floyd-Warshall and Johnson's, can be used to find shortest path in graphs with negative edges and cycles, each with their own characteristics and suitability for different types of graphs [14-16].

3.2 Implementation

This research's methodology included implementing 2 algorithms (Floyd-Warshall and Johnson's) on a Google Maps plotter using longitudes, latitudes and Google Maps API, using Python and GMplot library to create graphs and calculate shortest paths. The research team has also created a custom implementation of the algorithms to find the exact path of the shortest path calculation. The paper will evaluate the performance of the algorithms using runtime, memory usag and accuracy with test cases.

Table 1. Example of dataset.

ID	Loc (lat. & long.)	Type	Dest.	Dist.	Chain
0	-6.224494.., 106.804436..	street	1, 9	4.7, 4	3, 8, 21
1	-6.224072.., 106.804085..	street	2	4	3, 8
2	-6.223906.., 106.803957..	restaurant	3, 4	3.2, 3	3, 8

The data used to represent the nodes in the graph was stored in a csv file that was manually created by the team. The researchers also used the Google Maps API to obtain the latitude and longitude of various locations. These coordinates were then inputted into an Excel spreadsheet and saved in a CSVformat. The file included information about the identification of each node and its direct destination, as well as the distance from one node to another. Additionally, a column was included to store the latitude and longitude of each node as well as what type of node it was (e.g. street or restaurant) which will define the paths that the research

is interested in and actually want to show. Table 1 shown the example of the dataset in the research.

The researchers inputted several coordinates and locations in the Sudirman area around BINUS FX campus (Jakarta) and created several labels that gave users places to go for recreational activities, places to eat, malls and other universities in the area.

The research was done using Python by converting the csv file into an adjacency list and matrix for the research, which consisted of 500 vertices. This allowed for efficient storage and manipulation of the node data. In addition to finding the shortest path distance between two vertices, these modified Floyd-Warshall and Johnson's algorithms also return the exact path taken to calculate the distance. This is done by using a predecessor matrix, which is updated during the algorithm's execution and used to reconstruct the path. This modification can make the algorithm slower and use more memory, but it is important for real-life applications such as transportation systems, logistics, and telecommunication networks.

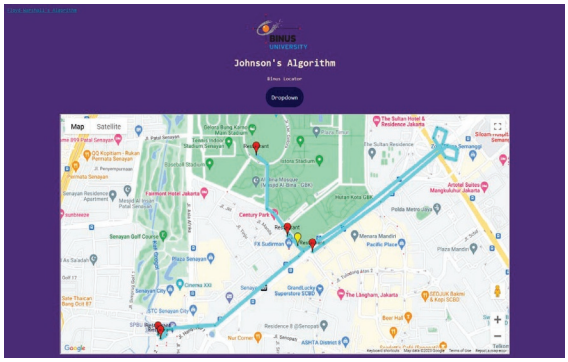

Fig. 5. The implementation.

4 Results and Discussions

The research tested the two algorithms over a series of different amount of nodes within a graph ranging from 10 to 500. The algorithm's mean (average) run time was measured (in seconds) and the memory usage of each algorithm was also tested. Both an adjacency list and adjacency matrix were used when conducting the measurements and thus the results of both will be shown. Table 2 and Table 3 show the results of the comparisons.

Table 2. Comparison of runtime and space complexity between Floyd-Warshall and Johnson algorithm on adjacency matrix

Input size (vertices)	Avg. Floyd-Warshall time (sec)	Floyd-Warshall space (byte)	Avg. Johnson time (sec)	Johnson space (byte)
10	0.005	184	0.0005	184
25	0.002	312	0.003	312
55	0.017	568	0.03	568
100	0.103	920	0.175	920
250	1.503	2200	2.912	2200
500	12.439	4216	20.024	4216

Table 3. Comparison of runtime and space complexity between Floyd-Warshall and Johnson algorithm on adjacency list.

Input size (vertices)	Avg. Floyd-Warshall time (sec)	Floyd-Warshall space (byte)	Avg. Johnson time (sec)	Johnson space (byte)
10	0.002	184	0.0005	184
25	0.004	312	0.001	312
55	0.027	568	0.011	568
100	0.109	920	0.05	920
250	1.58	2200	0.663	2200
500	12.415	4216	5.359	4216

The research study has found that the Floyd-Warshall algorithm had the best performance in terms of runtime and memory usage compared to Johnson's algorithm when using an adjacency matrix. However, Johnson's had better runtime for a smaller number of nodes and also performs better overal when using an adjacency list over matrix.

There are a few reasons why Johnson's algorithm might run much faster than Floyd-Warshall algorithm when the graph has a lot of vertices and is represented as an adjacency list. The time complexity of Johnson's algorithm is $O(VE + V^2 \log V)$, which is better than Floyd-Warshall algorithm's $O(V^3)$ for sparse graphs where E is the number of edges. Sparse graphs are just graphs that are not close to having the same amount of edges as vertices. Adjacency list representation is more memory efficient than an adjacency matrix representation when the graph is sparse, as it only stores the edges that actually exist, whereas an adjacency matrix would require storage for all potential edges.

Johnson's algorithm uses the Bellman-Ford algorithm to find the shortest path between all pairs of vertices, which is more efficient than Floyd-Warshall algorithm, especially when the graph is sparse, as it can stop early if it detects that there are no negative weight cycles. The second batch of graphs above show that the input representation (adjacency list) may not have a significant impact on the performance of the algorithm if the graph is sparse. In a sparse graph, the number of edges is much smaller than the number of nodes and thus the representation does not affect the algorithm's performance.

Continuing on as to why the Floyd-Warshall algorithm does not seem different on the differing input representations. An answer as to why the Floyd-Warshall algorithm might not show a significant difference in performance when comparing input as an adjacency list versus input as an adjacency matrix could be some of the following reasons.

The time complexity of the Floyd-Warshall algorithm is $O(V^3)$ regardless of the input representation. This means that the algorithm itself takes the same amount of time to run regardless of whether the input is an adjacency list or an adjacency matrix.

The space complexity of the Floyd-Warshall algorithm is also the same regardless of the input representation. The algorithm requires a VxV matrix to store the shortest path between all pairs of nodes. The reason for this is that the Floyd-Warshall algorithm

considers all possible paths between every pair of vertices, including paths that may pass through other intermediate vertices. In order to keep track of these different paths and the corresponding distances, a v x v matrix is used to store the shortest path tree (SPT). Each element in the matrix represents the shortest distance between a given pair of vertices, and can be updated as the algorithm progresses.

Additionally, the algorithm uses the matrix to check if there exists an edge from vertex i to vertex k and k to j. If it finds that the path from i to j passing through k is shorter than the present shortest path from i to j, it updates the shortest distance from i to j. This is done for all possible combinations of vertices i, j and k, which is why the algorithm requires a v x v matrix to store the SPT.

The number of edges in the graph does not affect the time complexity of the Floyd-Warshall algorithm. The algorithm looks at all possible paths between all pairs of nodes, which is why the time complexity is $O(V^3)$ regardless of the number of edges in the graph. The input representation may not have a significant impact on the performance of the algorithm if the graph is not very dense. In a sparse graph, the number of edges is much smaller than the number of nodes and thus the representation does not affect the algorithm's performance.

In summary, the Floyd-Warshall algorithm is not affected by the input representation (adjacency list or adjacency matrix) as its time and space complexity is the same regardless of the input representation and the input representation does not affect the algorithm's performance in sparse graph. It's always best to test both of these input representation on the graph that you are working on and choose the one that gives better results.

A two-sample t-test was conducted, with one sample being the values of the run-times of Johnson's algorithm that was used to calculate the average on an adjacency list and the other sample being the values of the run-time of Floyd's algorithm on an adjacency list. The t-value and p-value was calculated. A small p-value (typically less than 0.05) would indicate strong evidence against the null hypothesis and in favor of the alternative hypothesis, which would suggest that the runtime of the two algorithms is not the same. The equation below describes the t-test formula [17-19].

$$t = \frac{\bar{x}_1 - \bar{x}_2}{S_p\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}} \quad (1)$$

This study has shown that the Johnson's algorithm is the best choice for finding the shortest path between multiple locations in a Google Maps plotter using the Google Maps API when the graph is sparse and represented using an adjacency list. This is supported by the two-sample t-test, which showed that the p-value between the sample of Johnson's run-time and the sample of Floyd-Warshall's runtime is 2.776757786919472e-54, which is below the chosen alpha value (significance level) and allows us to reject the null hypothesis that the two algorithms have the same mean runtime.

Table 4. Results of T-test for the two runtimes of the algorithms.

	Floyd-Warshall	Johnson
Runtimes	12.86	5.51
	12.45	5.50
	12.63	5.35
	12.59	5.33
	12.25	5.43
	12.53	5.37
	12.25	5.24
	12.12	5.36
	12.25	5.42
	12.49	5.35
	12.54	5.37
	12.34	5.33
	12.13	5.17
	12.26	5.17
	12.27	5.28
	12.35	5.20
	12.33	5.19
	12.53	5.31
	12.29	5.18
	12.45	5.32
Statistics t-value	149.79	
p-value	2.78e-54	

The results of this research demonstrate that the Johnson's algorithm is faster and requires less memory than the Floyd-Warshall algorithm, which makes it a more suitable choice for finding the shortest path between multiple locations in a sparse graph when there is a large number of vertices. However, it is important to note that the Dijkstra's algorithm is still a suitable choice for small number of nodes and when there are no negative edge weights. Negative edge weights can be used to represent things such as beneficial routes or traffic and congestion. Therefore, it is essential to consider whether or not your application will need the use of negative edge weights in the future.

5 Conclusion and Recommendation

In this paper, the research presented a comparative study of the implementation of all-to-all pairs shortest path algorithms, specifically Floyd-Warshall and Johnson's. This paper demonstrated the use of these algorithms in finding the shortest path between multiple locations using a Google Maps plotter and the Google Maps API.

The research's results show that the Johnson's algorithm had the best performance in terms of runtime and memory usage when compared to Floyd-Warshall's algorithm. The research has also figured out that using an adjacency list is better when the graph is sparse as it usually is in real life than using an adjacency matrix. However, Floyd-Warshall still has a number of use cases as it is considerably faster when using an adjacency matrix as input which might be a hurdle that some have. It is also worth noting that the Floyd-Warshall algorithm is generally much easier to implement than the Johnson's algorithm and can cut down on time if the time it takes for the algorithm to complete is not important, for example if the number of vertices in the graph stay small then the Floyd-Warshall

algorithm might be a better idea to use instead even if the Johnson's algorithm is slightly faster.

Based on the findings of this study, this paper recommends the use of these algorithms in optimizing routes for transportation and logistics. For example, the algorithms can be used to find the shortest path between multiple locations, which can be used to plan efficient routes for delivery vehicles. It can also be used to deal with specific constraints such as traffic.

Further research can be done to improve the scalability and performance of the algorithms. Additionally, research can be done to explore the use of these algorithms in other real-world scenarios, such as in the case of social networks or in the field of computer science.

Overall, the use of the Floyd-Warshall and Johnson's algorithms in real-world scenarios has the potential to greatly improve efficiency and optimize various processes. Researchers and practitioners can easily apply these algorithms in a realworld setting and gain valuable insights.

References

1. H. Abu-Ryash, A. Tamimi, *Comparison Studies for Different Shortest path Algorithms*, Int. J. Comp. Appl. **14**, 8, 5979-5986 (2015)
2. I. K. Laga Dwi Pandika, B. Irawan, C. Setianingsih, *Application of optimization heavy traffic path with floyd-warshall algorithm*, in Proceedings of the International Conference on Control, Electronics, Renewable Energy and Communications, ICCEREC, 5-7 Dec 2018, Bandung, Indonesia (2018)
3. B. Amaliah, C. Fatichah, O. Riptianingdyah, *Finding the Shortest Paths Among Cities in Java Island Using Node Combination Based on Dijkstra Algorithm*, Int. J. Smart Sens. Intell. Syst. **9**, 4, 2219-2236 (2016)
4. H. Mahmoud, N. Akkari, *Shortest path calculation: a comparative study for location-based recommender system*, in in 2016 world symposium on computer applications & research, WSCAR, 12-14 Mar 2016, Cairo, Egypt (2016)
5. M. Afser, M. U. Shameem, J. Ferdous, M. Milon, M. Hossain, *Study on single source shortest path algorithms*, Ph.D. dissertation, United International University (2017)
6. N. Biggs, E. K. Lloyd, R. J. Wilson, *Graph theory*, Oxford University Press, 1736-1936 (1986)
7. H. Singh, R. Sharma, Int. J. Comp. Tech. **3**, 1, 179-183 (2012)
8. X. Yang, *Adjacent matrix,* https://www.cs.mtsu.edu/~xyang/3080/adjacencyMatrix.html (n.d)
9. GeeksforGeeks, *Graphs and its representations*, https://www.geeksforgeeks.org/graph-and-its-representations/ (n.d)
10. X. Huang, *Negative-weight cycle algorithms*, in FCS, pp. 109-115 (2006)
11. G. Melancon, *Just how dense are dense graphs in the real world? a methodological note*, in Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization, AVI06, 23 May 2006, Venice, Italy (2006)
12. Math.stackexchange, *Maximally dense unit distance graphs*, https://math.stackexchange.com/questions/2575268/maximally-dense-unit-distance-graphs (2019)
13. Stackoverflow, *Find center of a sparse graph*, https://stackoverflow.com/questions/18285998/find-center-of-a-sparse-graph (2013)
14. R. Risald, A. E. Mirino, S. Suyoto, *Best routes selection using Dijkstra and Floyd-Warshall algorithm*, in 2017 11th International Conference on Information & Communication Technology and System, ICTS, pp. 155-158 (2017)
15. M. A. Samosir, Formosa J. Sci. Tech. **2**, 2, 453-474 (2023)
16. R. Johner, A. Lanaia, R. Dornberger, T. Hanne, *Comparing the Pathfinding Algorithms A, Dijkstra's, Bellman-Ford, Floyd-Warshall, and Best First Search for the Paparazzi Problem*, in Congress on Intelligent Systems: Proceedings of CIS 2021, **2**, pp. 561-576 (2022)
17. T. K. Kim, *T test as a parametric statistic Korean*, J. Anesthesiol. **68**, 6, 540-546 (2015)
18. T. K. Kim, J. H. Park*, More about the basic assumptions of t-test: normality and sample size,* Korean J. Anesthesiol, **72**, 4, 331-335 (2019)
19. P. Mishra, U. Singh, C. M. Pandey, P. Mishra, G. Pandey, Application of Student's *t*-test, Analysis of Variance, and Covariance, Anaesth. **22**, 4, 407 (2019)