RESEARCH ARTICLE | JULY 21 2023

Integrated implementation of fuzzy logic and Dijkstra's algorithm in travel routes planning FREE

Murni ✉; Tri Handhika

AIP Conference Proceedings 2689, 130004 (2023)
https://doi.org/10.1063/5.0119008

Integrated Implementation of Fuzzy Logic and Dijkstra's Algorithm in Travel Routes Planning

Murni[2, a)] and Tri Handhika[1, b)]

[1]*Centre for Computational Mathematics Studies, Gunadarma University, 100 Margonda Raya Street, Pondok Cina, Depok, West Java, 16424, Indonesia*
[2]*Department of Informatics Engineering, Faculty of Industrial Technology, Gunadarma University, 100 Margonda Raya Street, Pondok Cina, Depok, West Java, 16424, Indonesia*

[a)]*Corresponding author: murnipskm@staff.gunadarma.ac.id*
[b)]*trihandika@staff.gunadarma.ac.id*

Abstract. This paper addressed the fuzzy logic and integrated Dijkstra's algorithm to determine an efficient path for travel routes planning. Dijkstra's algorithm is the same as Breadth-First Search (BFS) algorithm, i.e. queuing principle, but the queue in Dijkstra's Algorithm is a priority queue. The research methodology consists of data collection, clustering, fuzzy logic modelling, implementation of the Dijkstra algorithm, and testing the results. The collected data in this paper is the distance and travel time. Data clustering is needed to classify time data based on congestion level because travel time data is dynamic. The results of the clustering become a reference for selecting the time in collecting travel time data. The time represents the current state of the traffic. Fuzzy logic modelling is done after all the required data has been collected. This paper uses the Tsukamoto fuzzy method because this method has tolerance for data and is very flexible. The data is developed into a mathematical model to produce a crisp output (i.e. travel weight). The crisp output is then processed with Dijkstra's algorithm to produce a solution that solves the problem in this paper. The final result of this research is that fuzzy logic and integrated Dijkstra's algorithm is ideal for efficient pathfinding because it always gives the most efficient results from all possible paths available with the existing problem constraints.

INTRODUCTION

Information and Communication Technology (ICT) is currently developing very rapidly. ICT in its use can present various kinds of information, for example, information about traffic conditions. Information on traffic conditions can be in the form of traffic flow density and road distance. This information can be used in various ways, one of which is travel activities. Knowing information about traffic conditions can be used as an indicator in the search for efficient distribution channels. Therefore, the use of ICT in travel activities is needed.

Several methods can solve the problem of finding an efficient distribution channel. For example, the Floyd-Warshall method calculates the smallest weight of all paths that connect a pair of points and does it at once for all points [1]. In addition, the Bellman-ford method works by calculating all the shortest distances starting from one node [2]. Then there is the A-star method using the distance plus cost heuristic function to determine the order in which the search is carried out through the nodes in the tree [3]. Finally, the Dijkstra method works using the greedy principle, where at each step, the minimum weight is selected that connects a node that has been selected with another node that has not been selected.

This study uses the Dijkstra method because the Dijkstra method is more profitable in terms of run time than the Bellman-ford method as long as the graph does not contain negative vertices [4]. Then, Dijkstra's method of route finding process time is faster than the Floyd-Warshall and A-star methods [5]. In addition, Dijkstra's method operates thoroughly against all available alternatives so that the shortest path can be obtained from all nodes [6]. In this study, Dijkstra's method is represented in an algorithm to be run by the ICT.

Toward Adaptive Research and Technology Development for Future Life
AIP Conf. Proc. 2689, 130004-1–130004-8; https://doi.org/10.1063/5.0119008
Published by AIP Publishing. 978-0-7354-4470-6/$30.00

The efficiency of the distribution channel referred to in this paper is the distribution channel with the smallest value of travel weight. The value of the trip weight is determined by two parameters, namely the distance of the road and the travel time. The use of more than one parameter, especially in multimodal networks, makes pathfinding algorithms, such as Dijkstra's algorithm, unable to run optimally [7].

Thus, this study discusses the integrated implementation of fuzzy logic and Dijkstra's algorithm to search for efficient distribution paths. Integrated into this case is fuzzy logic that gives output to Dijkstra's algorithm in the form of a collection of road weight values used to determine an efficient distribution path.

FUZZY LOGIC

Fuzzy logic is an appropriate way to map an input space into an output space [8]. Fuzzy logic works by using the degree of membership of a value which is then used to determine the results produced based on predetermined specifications. Several things need to be known in understanding fuzzy logic, i.e. Universe, domain, and fuzzy set.

A fuzzy set is a group that represents a certain condition or situation in a fuzzy variable. The elements of the fuzzy set lie in the range 0 to 1. A fuzzy set is defined as a set characterized by a function denoted by μ. The membership function is a curve that shows the mapping of data input points into their membership values (often also called membership degrees) which has an interval between 0 to 1 [9]. Several types of membership functions can be used, i.e. Linear Representation. The input mapping to the degree of membership is represented as a straight line in a linear representation. There are two states of linear fuzzy sets, ascending linear representation and descending linear representation.

Membership function ascending linear representation:

$$\mu\,[x] \begin{cases} 0 & ; \quad x \leq a \\ \frac{\sum \alpha i z i}{\sum \alpha i} & ; \quad a < x < b \\ 1 & ; \quad x \geq b \end{cases} \tag{1}$$

Membership function descending linear representation:

$$\mu\,[x] \begin{cases} 1 & ; \quad x \leq a \\ \frac{b-x}{b-a} & ; \quad a < x < \\ 0 & : \quad x \geq b \end{cases} \tag{2}$$

where:$x$ismembership value, a is the lowest member score, and b is the highest member score.

Fuzzy logic has three methods, namely the fuzzy Tsukamoto, Sugeno, and Mamdani methods. This paper uses the Tsukamoto fuzzy method because this method has tolerance for data and is very flexible [10]. The advantage of the Tsukamoto method is that it is intuitive and can provide responses based on qualitative, inaccurate, and ambiguous [11]. In Tsukamoto's fuzzy method, every consequence of the rule in the form of If-then must be represented by a fuzzy set with a monotonous membership function as a result (fuzzification process). The output of the inference results from each rule is given explicitly (crisp) based on the α-predicate (the element value as a result of the operation of 2 sets is often known as fire strength). The final result was obtained using a weighted mean defuzzification [12]. In general, there are three steps for the Tsukamoto fuzzy method, such as Fuzzification, Inference, and Defuzzification.

The first stage in the fuzzy calculation is Fuzzification, which changes the crisp value to a fuzzy value. Where x is the definition of the variable from the fuzzy set vector, the fuzzifier is the definition of changing the crisp value (crisp) to the fuzzy set, and 0 is a vector of firm values of an input variable [12].

A fuzzy inference system concludes fuzzy rules or rules based on fuzzy set theory, fuzzy rules in the form of if-then, and reasoning with input and output in the form of crisp values. When evaluating the rules in the inference engine, Tsukamoto's fuzzy method uses the MIN implication function to get the α-predicate value of each rule (α_1, α_2, α_3, …., α_n). The formula to find the value of α-predicate is written in Equation (3).

$$\alpha_{predicate} = \mu_{(x)} \cap \mu_{(y)} \quad (3)$$

where: $\mu_{(x)}$ and $\mu_{(y)}$ is the element degree of the input variable. Each value of α - predicate is used to calculate the inference result explicitly (crisp) in each rule (z_1, z_2, z_3, ..., z_n).

Defuzzification is converting the fuzzy output into a crisp value according to the specified membership function. In this study using the Weighted Average method, the defuzzification process is different from the previous one. This process can only be used if the output membership functions of several fuzzy processes have the same form. This method is represented in the formula in Equation (4) [11].

$$\frac{\sum \alpha_i \, z_i}{\sum \alpha_i} \quad (4)$$

where: Z is defuzzification, α_i is alpha predicate rule – i, and z_i is inference output.

Fuzzy logic modelling is done after all the required data has been collected. The collected data in this paper is the distance and travel time. Travel time data is dynamic because different times can change their values. Therefore, data clustering is needed to classify time data based on congestion level. The results of the clustering become a reference for selecting the time in collecting travel time data. The time represents the current state of the traffic. The data is developed into a mathematical model to produce a crisp output (i.e. travel weight). The crisp output is then processed with Dijkstra's algorithm to produce a solution that solves the problem in this paper.

DJIKSTRA'S ALGORITHM

Dijkstra's algorithm is the same as Breadth-First Search (BFS) algorithm, i.e. queuing principle, but the queue in Dijkstra's Algorithm is a priority queue. This priority queue is used in Dijkstra's algorithm to determine only the highest priority vertices to be traced. Dijkstra's algorithm is used in this paper to find an efficient path that connects two points on a directed graph. The efficient path in this paper is the path with the smallest weight that connects the origin and destination points. The logical sequence of Dijkstra's Algorithm is the basis for making programs to run this algorithm by Information and Communication Technology (ICT) devices. Djikstra's algorithm is presented as a flowchart in Fig. 1 [13].

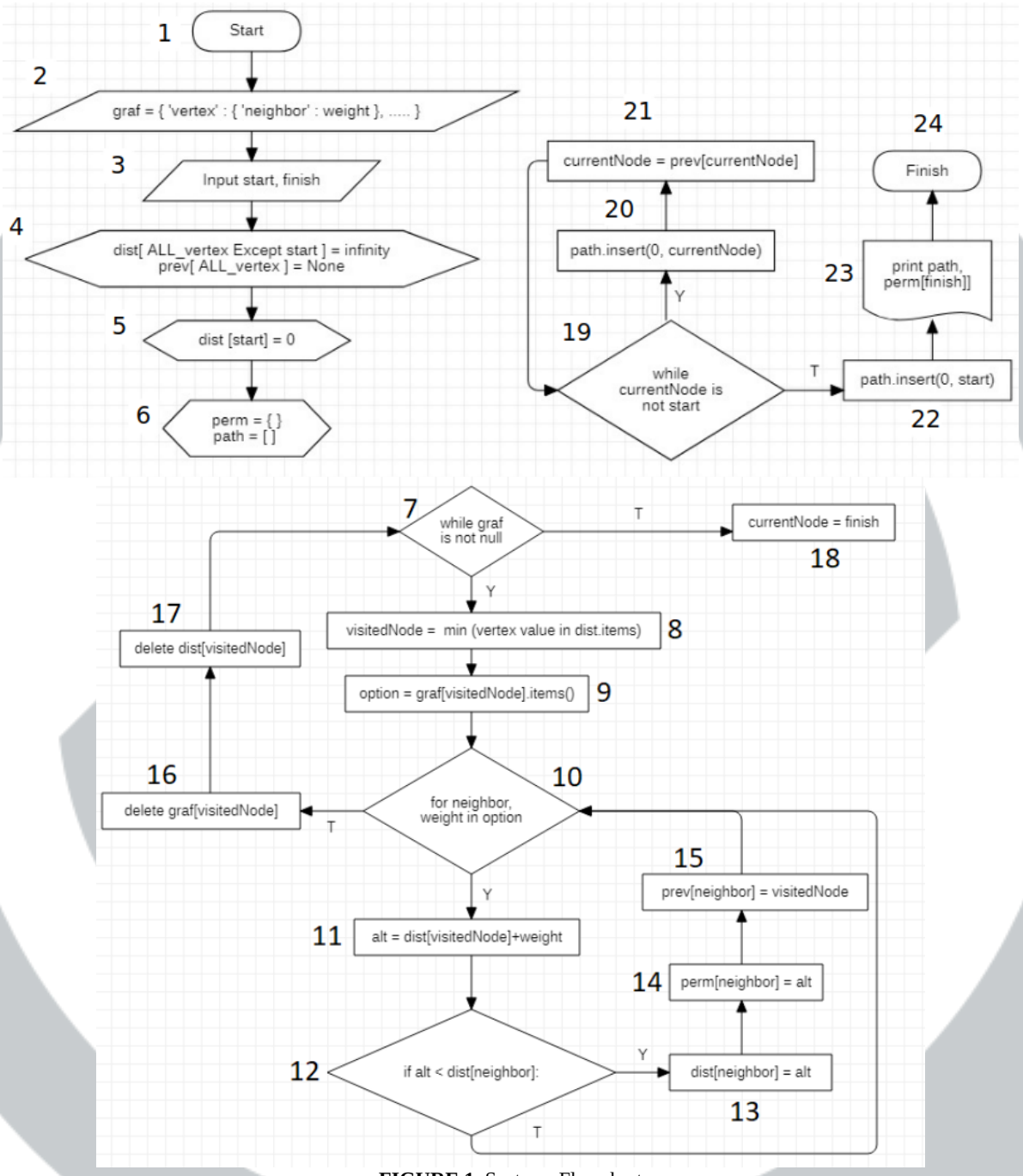

FIGURE 1. Systems Flowchart

The explanation for the flowchart of Dijkstra's Algorithm is adjusted to the following numbers:

1. The program is executed.
2. The data is entered from the path graph about the existing vertices represented by the vertex variable. The neighbour variable represents the vertex adjacent to the vertex. The weight variable represents the weight that connects the two points.
3. Other input data is that the start variable represents the starting point, and the destination point is represented by the finished variable.
4. The dist variable is declared to store the weight data for each existing vertex. The initial initiation value in the dist variable for all vertices except the starting point is infinity or a very large value. The prev variable is

declared to store previous vertex data to reach that point. The data is useful in determining an efficient path from the starting point to the destination point.

5. The initial initiation value in the dist variable for the starting point is 0 because the weight to the same point is 0. This variable determines the point visited/checked.
6. The perm variable is declared to store the weight values for each existing vertex. The difference with the dist variable is that the value in the perm variable becomes the result of the calculation of the weight of the Dijkstra algorithm/stores the final value. Path variable to store efficient distribution path.
7. After all the necessary variables are available, then go to the initial iteration stage. The initial loop has a condition: as long as the value in the graph still exists, the loop continues. If the conditions are met, the program will go to number 7 and if not met, go to number 18.
8. First, determine the visited/checked vertices. The selected vertex is the vertex with the smallest value of the dist variable. For example, if this process has only been executed once, the selected vertex is started. It is because all the vertices in the dist variable are infinity other than the start. The selected vertex becomes the value of the visitedNode variable.
9. Next, look for the connected vertex/neighbour with visitedNode. It is done by retrieving the item value from the visitedNode on the graph variable. The value of the connected vertex becomes the value of the option variable.
10. Then check all the values in the options variable, namely neighbour and weight. The test uses iteration so that the loop is in a nested loop. If the loop still occurs, the program goes to number 11, and if the loop has finished, the program goes to number 16.
11. First, the available alternative weights are calculated. The alternative weight value is obtained from the visitedNode values on the dist and weight variables. The alternative weight value is stored in the alt variable.
12. The value of the alt variable is then analyzed by branching. The branch has the condition that the value of the alt variable < the value of the dist neighbour on the dist variable. If the condition fulfils, go to number 13. If the condition does not meet, return to the beginning of the nested loop, which is number 10 to check the value of another option variable.
13. The neighbour value in the dist variable is updated with the alt variable value.
14. The neighbour value in the perm variable is updated with the alt variable value.
15. The neighbour value in the prev variable is updated to the visitedNode value. Then, return to the beginning of the nested loop, which is number 10 to check the value of another option variable.
16. After the nested loop is finished, the visitedNode value in the graph variable is deleted.
17. Then deleted the visitedNode value in the dist variable, indicating that the vertex has been visited/checked. The process in this initial loop continues until the value of the graph variable is null. The next step is to determine an efficient path based on weights. This process uses prev and path variables.
18. The currentNode variable is declared to be the starting point for efficient path tracking. The initial initialization value of the currentNode variable is finished.
19. Then enter the final loop stage. The final loop has a condition: as long as the currentNode variable does not start, the loop continues. If these conditions are met, the program will go to number 20. If not, it will go to number 22.
20. The value of the currentNode variable is stored at the beginning of the list of path variable values.
21. The currentNode variable value is updated with the currentNode value in the prev variable, which is the previous vertex of currentNode. Then the program returns to number 19.
22. After exiting the final loop, the currentNode variable value is stored at the beginning of the path variable value list.
23. The last stage is the path variable is printed as an efficient path, and the finish value on the path variable is the travel weight value from start to finish.
24. Program completed.
25. Dijkstra's algorithm results are then tested to prove that the results are in line with expectations.

IMPLEMENTATION

In this paper, we choose 15 attractions in this research to demonstrate how advanced applications with the proposed model make travel itinerary recommendations. The road distance and travel time data are then represented in the form of a graph as follows (Fig. 2):

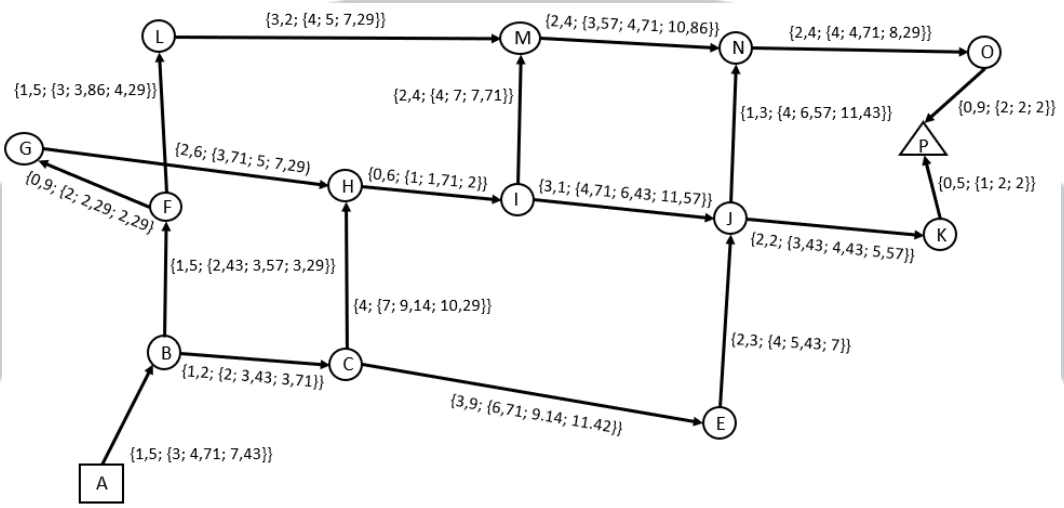

FIGURE 2. Graph of distance and travel time of each edge

Fuzzification, inference, and defuzzification steps are carried out for all edges each time to obtain crisp output for each edge (Figure 3).

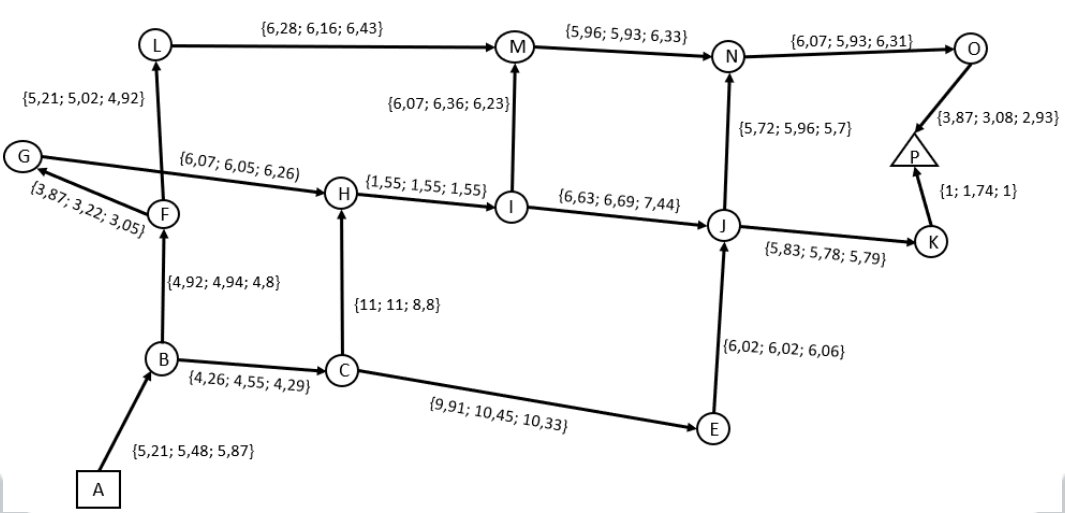

FIGURE 3. Crisp output (i.e. travel weight)

Dijkstra's algorithm is used in this paper to find an efficient path that connects two points on a directed graph. The efficient path in this paper is the path with the smallest weight that connects the origin and the point. The results of Dijkstra's algorithm [14] are tested to ensure the path and the resulting weights are efficient. The results were tested by comparing the total weights of all possible paths to the destination point from the starting point. Test results are carried out at all times. In this paper, the systems flowchart (Fig. 1) is written using python programming.

- Free Period Path Test

TABLE 1. Travel weight for free time

Possible Paths	Weight
A – B – C – E – J – K – P	32,23
A – B – C – E – J – N – O – P	35,34
A – B – C – H – I – J – K – P	35,48

Possible Paths	Weight
A – B – C – H – I – M – N – O – P	42,44
A – B – C – H – I – J – N – O – P	38,59
A – B – F – G – H – I – J – K – P	35,08
A – B – F – G – H – I – M – N – O – P	37,52
A – B – F – G – H – I – J – N – O – P	43,91
A – B – F – L – M – N – O – P	37,52

Based on the data in Table 1, it can be concluded that the efficient path and weight in the free time is the path A - B - C - E - J - K - P with a weight of 32.23 because the weight of 32.23 is the smallest weight of all possibilities. These results are then compared with the output of Dijkstra's algorithm. The output of Dijkstra's algorithm for the free time and the results of Table 1 produce the same output. So it can be concluded that fuzzy logic and the integrated Dijkstra algorithm produce efficient paths and weights in the free time.

- Normal Time Path Test

TABLE 2. Normal time travel weight

Possible Paths	Weight
A – B – C – E – J – K – P	34,02
A – B – C – E – J – N – O – P	35,51
A – B – C – H – I – J – K – P	36,79
A – B – C – H – I – M – N – O – P	42,33
A – B – C – H – I – J – N – O – P	38,28
A – B – F – G – H – I – J – K – P	35,45
A – B – F – G – H – I – M – N – O – P	36,61
A – B – F – G – H – I – J – N – O – P	42,9
A – B – F – L – M – N – O – P	36,54

Based on the data in Table 2, it can be concluded that the efficient path and weights in the normal time are paths A – B – C – E – J – K – P with a weight of 34.02 because of the weight of 34.02 is the smallest weight of all possibilities. These results are then compared with the output of Dijkstra's algorithm. The output of Dijkstra's normal time algorithm and the results of Table 2 produce the same output. So it can be concluded that fuzzy logic and the integrated Dijkstra algorithm produce efficient paths and weights in the normal time.

- Traffic Time Path Test

TABLE 3. Traffic time travel weight

Possible Paths	Weight
A – B – C – E – J – K – P	33,34
A – B – C – E – J – N – O – P	35,79
A – B – C – H – I – J – K – P	34,74
A – B – C – H – I – M – N – O – P	40,76
A – B – C – H – I – J – N – O – P	37,19
A – B – F – G – H – I – J – K – P	35,76
A – B – F – G – H – I – M – N – O – P	37,02
A – B – F – G – H – I – J – N – O – P	43,91
A – B – F – L – M – N – O – P	37,59

Based on the data in Table 3, it can be concluded that the efficient path and weights in the solid time are the paths A – B – C – E – J – K – P with a weight of 33.34 because the weight of 33.34 is the smallest weight of all possibilities. These results are then compared with the output of Dijkstra's algorithm. The output of Dijkstra's algorithm for solid time and the results of Table 3 produce the same output. So it can be concluded that fuzzy logic and the integrated Dijkstra algorithm produce efficient paths and weights in a solid time.

CONCLUSION

Based on the discussion of this paper, it can be concluded that Fuzzy logic and integrated Dijkstra algorithm can get an efficient path, namely the path with the smallest weight from the starting point to the destination point. The output of fuzzy logic and integrated Dijkstra's algorithm for all time is the same as the efficient distribution path in manual calculations by checking the path with the smallest integrated weight of all possible paths from the starting point to the destination point. The outputs tested are paths and total weights. Therefore, fuzzy logic and integrated Dijkstra's algorithm are ideal for efficient pathfinding because it always gives the most efficient result from all possible paths available with the given problem constraints.

This result is obtained by using arbitrary data. If we depart from point A, the efficient path is **A – B – C – E – J – K – P**.

As future work, the addition of variable characteristics of the road to determine the weight of each trip segment on the graph is possible. The addition of these variables makes the weights on each segment better at representing the state of the road.

ACKNOWLEDGMENT

This research was funded by PTUPT Research Grant 2021, Direktorat Riset dan Pengabdian Masyarakat, Direktorat Jenderal Penguatan Riset dan Pengembangan Kementerian Riset, Teknologi, dan Pendidikan Tinggi (DRPM KEMENRISTEKDIKTI) Indonesia to Gunadarma University No. 234/SP2H/LT/DRPM/2021.

REFERENCES

1. Ramadhan Z, Zarlis M, Efendi S, Siahaan APU. Perbandingan Algoritma Prim dengan Algoritma Floyd-Warshall dalam Menentukan Rute Terpendek (Shortest Path Problem). JURIKOM (Jurnal Ris Komputer). 2018;5(2):135–9.
2. Hutasoit ETH. Pencarian Rute Terpendek Menggunakan Algoritma Bellman-Ford (Studi Kasus: PT. JNE Medan). J Sist Komput dan Inform. 2019;1(1):20–5.
3. Dalem IBGWA. Penerapan algoritma A*(Star) menggunakan graph untuk menghitung jarak terpendek. J Resist (Rekayasa Sist Komputer). 2018;1(1):41–7.
4. Prasetyo BIA, Maslan A. Analisis Perbandingan Pada Algoritma Bellman Ford Dan Dijkstra Pada Google Map. Khazanah Ilmu Berazam. 2020;3(2):337–49.
5. Umar R, Yudhana A, Prayudi A. Analisis Perbandingan Algoritma Djikstra, A-Star, dan Floyd Warshall dalam Pencarian Rute Terdekat pada Objek Wisata Kabupaten Dompu. J Teknol Inf dan Ilmu Komput. 2021;8(2):227–34.
6. Lubis HS. Perbandingan Algoritma Greedy dan Dijkstra untuk menentukan lintasan terpendek. Dep Mat Univ Sumatera Utara, Medan. 2009;
7. Golnarkar A, Alesheikh AA, Malek MR. Solving best path problem on multimodal transportation networks with fuzzy costs. Iran J Fuzzy Syst. 2010;7(3):1–13.
8. Jang J-SR, Sun C-T, Mizutani E. Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [Book Review]. IEEE Trans Automat Contr. 1997;42(10):1482–4.
9. Wang L-X. A course in fuzzy systems and control. Prentice-Hall, Inc.; 1997.
10. Sari NR, Mahmudy WF. Fuzzy inference system Tsukamoto untuk menentukan kelayakan calon pegawai. SESINDO 2015. 2015;2015.
11. Maryaningsih M, Siswanto S, Mesterjon M. Metode Logika Fuzzy Tsukamoto Dalam Sistem Pengambilan Keputusan Penerimaan Beasiswa. J Media Infotama. 2013;9(1).
12. Ross TJ. Fuzzy logic with engineering applications. John Wiley & Sons; 2004.
13. Deng Y, Chen Y, Zhang Y, Mahadevan S. Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment. Appl Soft Comput. 2012;12(3):1231–7.
14. Lee W-M. Python machine learning. John Wiley & Sons; 2019.