Research on the Selection of Path Planning Algorithm: A Case Study in Leeds

To cite this article: Pang Yan 2022 *J. Phys.: Conf. Ser.* **2179** 012039

Research on the Selection of Path Planning Algorithm: A Case Study in Leeds

Pang Yan*

University of Leeds, Southwest Jiaotong University, Sichuan Chengdu 610031, China

*Corresponding author email: sc18y2p@leeds.ac.uk

Abstract. Path planning algorithms are the core of path planning technology, and different path planning algorithms have different performances in terms of search efficiency and planning efficiency in different application areas. The traditional path planning algorithm selection mainly relies on personal knowledge and experience and takes a qualitative analysis method. With the variety of path planning algorithms and the increase of environmental complexity, the limitations of this empirical method are more and more obvious, and it is difficult to meet the actual needs of path planning and achieve the ideal path planning effect. The selection of path planning algorithms is an evaluation problem involving multiple factors and requires comprehensive algorithm selection and analysis. This paper addresses the problems of the selection method of the path planning algorithm, applies uncertainty theory and multi-attribute decision theory, constructs an evaluation system for path planning algorithm selection, proposes a multi-criteria EV-VIKOR decision model and decision method for path planning algorithm selection, and combines the results show the effectiveness of the decision-making method proposed in this paper.

1. Introduction

With the development of information technology and artificial intelligence technology, many new industries and application areas have arisen along with them, such as unmanned vehicles, mobile robots, unmanned aerial vehicles (boats), GPS navigation, communication routing problems, takeaway delivery, fire escape, etc. And these industries and applications all involve a common problem that needs to be solved, which is the path planning problem, so path planning algorithms have attracted the attention and interest of many research scholars. In recent years, a series of path optimization algorithms, such as the A* algorithm and Floyd's algorithm, have been proposed for different application areas. However, these algorithms are not perfect and have their strengths and weaknesses in terms of search time, path planning efficiency, algorithm time complexity, space complexity, and directivity. In the past, people only chose algorithms based on their personal experience and knowledge, and the results of such choices would be difficult to achieve the desired results. To select path planning algorithms with better performance in all aspects, it is necessary to compare these algorithms and to compare the advantages and disadvantages of various algorithms, it is necessary to conduct a comprehensive analysis of multiple index factors, which is a multi-criteria decision analysis problem.

For a long time, the research related to path planning algorithms focuses on the improvement and optimization of algorithms and is dedicated to proposing better algorithms. Most of the literature only focuses on and pursues the improvement and optimization of algorithms in terms of individual metrics such as search performance, while there is little discussion on the comparison of the differences in overall effectiveness between algorithms and how to select algorithms for path search and planning.

Some scholars have only done some review analysis on the analysis of the characteristics and comparative selection of various path planning algorithms. For example, Li et al.[1] reviewed traditional algorithms, intelligent optimization algorithms, reinforcement learning-based algorithms, and hybrid path planning algorithms for unmanned vehicles, and compared and analyzed the performance of each algorithm in terms of real-time, robustness, and time complexity of the algorithms. Huo et al.[2] divided the mobile robot path planning into two categories, global planning and local planning, summarized the advantages and disadvantages of the relevant algorithms. Gu et al.[3] conducted a comparative analysis of the genetic algorithm, the RRT algorithm, and the A* algorithm, pointing out the shortcomings and deficiencies of these three algorithms, and improved them according to the specific problem to be solved. Zhao et al.[4] systematically introduced the algorithmic principles and algorithmic processes of Dijkstra's algorithm, Floyd's algorithm, Bellman-Ford's algorithm, and SPFA's algorithm, and conducted a comprehensive comparative analysis of the time complexity and applicability of these path planning algorithms. Liang et al.[5]divided path planning algorithms into evolutionary and non-evolutionary algorithms and subdivided the non-evolutionary algorithms into two categories based on mathematical features: classical mathematics and geometric graph theory. Chen et al.[6]divided the UAV path planning algorithms into classical algorithms and intelligent reactive algorithms. They also analyzed the basic principles and problems of various algorithms in depth. Yang et al.[7]classified path planning algorithms into exact methods, heuristic algorithms and intelligent optimization algorithms. Zhang et al.[8] classified the unmanned vehicle path planning algorithms into two categories, global planning and local planning.
However, the aforementioned research on the selection of path planning algorithms is still lacking systematically, and only qualitative comparisons are made on the performance of various path algorithms, as well as on the improvement of the algorithms and other aspects. This paper addresses the shortcomings of path planning algorithm selection and proposes the EV-VIKOR multi-attribute decision method for path planning algorithm selection based on uncertainty multi-attribute decision theory.

2. Evaluation Criteria

2.1. Time Complexity
The essence of algorithm time complexity is the execution time of the algorithm, i.e. the sum of the frequencies of all statements in the algorithm. The statement frequency is the number of times a statement is executed and is closely related to the size of the problem that the algorithm is solving. Assuming that for a given algorithm, the current problem size is n, the statement frequency can be expressed as a function $T(n)$ of the problem size, and the algorithm time complexity can then be expressed as $T(n)$.
Common time complexity scales are constant order $O(1)$, logarithmic order $O(logn)$, linear order O(n), linear logarithmic order $O(nlogn)$, square order $O(n^2)$, cubic order $O(n^3)$, kth order $O(n^k)$, and exponential order $O(2^n)$.

2.2. Space Complexity
Space complexity is a measure of the amount of temporary storage space occupied by an algorithm during its run, denoted as $S(n) = O(f(n))$. The amount of temporary storage space occupied during runtime varies from path algorithm to path algorithm. Some algorithms require only a small amount of temporary space and do not change with the size of the problem, so they are said to be "in-situ" and are storage-efficient. Some algorithms require several temporary work units that are related to the size of the problem, n, and increases with n. When n is large, more storage units will be used.

2.3. Search Efficiency
Global path planning algorithms plan paths based on a priori information, while local path planning algorithms plan by acquiring information about the environment through a perceptron. Regardless of the algorithm, it must have the ability to respond quickly to environmental information, have a low

computational effort and a short search time, and have good search efficiency. Different algorithms perform differently in terms of search efficiency. For example, the A* algorithm has the advantage of responding quickly to the environment and is a straightforward search algorithm.

2.4. Adaptability
There are many different path planning algorithms, and the range of adaptation varies from algorithm to algorithm due to the design of their algorithm principles and their computational performance. If the algorithm can adapt to various situations such as dense or dense relaxed graphs, graphs with positive or negative edge weights, and shortest paths with single or multiple sources, the wider the adaptation range, the better the algorithm. For example, Floyd's algorithm is suitable for multi-source shortest paths, which can calculate the shortest distance between any two nodes, is suitable for dense graphs, is more closely related to the vertices, and can solve the problem of negative-weighted edges, but the time complexity is relatively high and is not suitable for calculating a large amount of data.

3. Path Planning Algorithms

3.1. Depth-first Search(DFS)
Depth-first search is a graph traversal algorithm proposed by Hopcroft and Robert Tarjan. The algorithm prioritizes the expansion of the most recently discovered node until a certain depth limit is reached. If the target is not found or cannot be extended any further, it is then backtracked to another node to continue the extension.
Algorithm steps:
step1. Push the root node into the stack.
step2. Take the element from the top of the stack and find all the nodes in the adjacency table that are connected to him and have not been visited. Push these nodes into the stack. Mark the current node as the predecessor node of these nodes.
step3. Find the target node and exit.
step4. Traverse all nodes without finding the target node and exit the program.
Compared to Dijkstra's algorithm and Floyd's algorithm, depth-first search and breadth-first search are more oriented towards graph traversal operations, and the two algorithms are different forms of graph traversal and searching for shortest paths can be achieved by both algorithms, but their performance and efficiency are inferior to those of Dijkstra's algorithm and Floyd's algorithm and the A* algorithm. Depth-first search can find paths, but there is a great deal of uncertainty as to whether the path is the shortest. There is no perfect solution for the selection of the next node of the current node because of the choice of weights.

3.2. Breadth-first Search(BFS)
The breadth-first search was proposed by Edward Moore in 1959. The algorithm is an algorithm for traversing or searching a tree or graph data structure, starting at the root of the tree or some arbitrary node of the graph and exploring all neighboring nodes at the current depth before moving to the next node at the next depth level.
Algorithm steps:
step1. Add the root node to the queue.
step2. Each time you pop out the head element of the queue, look up all the nodes adjacent to it in the adjacency table and add the unvisited node to the queue. And set this node as the precursor of all new nodes added to the queue.
step3. Find the target node and exit.
step4. Iterate through all nodes without finding the target node, exit the program.
Dijkstra's algorithm is an improvement over breadth-first search in that for each node, the optimal solution is guaranteed to have been reached. Then, at the moment when the target node is finally found, the optimal solution must also be found. But the program will take longer to run.

3.3. Dijkstra Algorithm
Dijkstra algorithm[9] was proposed by the Dutch scientist Edsger W. Dijkstra in 1959. The key to this algorithm is that after each finding of a node that has the shortest path and has not been visited, the path is updated for all nodes to ensure that the optimal solution is obtained.
Algorithm steps:

step1. Create an array to mark whether a node is visited or not.

step2. Create a node to store the temporary distance of each node to the starting point. You can set the initial node to 0 and the other nodes to infinity. Set the current node as the start node.

step3. Set the current node to be accessible, e.g. mark it as "1" in the identity array. The array marked with "1" will not be accessed in the future.

step4. For the current node, iterate through all unvisited neighboring nodes and calculate a new distance from the current node. Compare the new distance with the value in the distance array and update or keep it as the smaller value. If the minimum value of the distance in the distance array is infinity, or if the endpoint has been marked as visited then stop. The procedure ends, otherwise, go to the next step.

Step 5. Continue to select the unvisited node marked with the minimum distance from the distance as the current node, continuing the loop from step 3.

In comparison with other algorithms, Dijkstra's algorithm is more accurate for shortest circuit search than the two graph traversal algorithms of DFS and BFS algorithms. In contrast to Floyd's algorithm, Dijkstra's time complexity is significantly better than Floyd's $O(n^3)$, but Floyd's algorithm is easier to understand and simpler to write in terms of program implementation.

3.4. Floyd Algorithm
Floyd algorithm[10] was proposed by Turing Award-winning professor Robert W. Floyd in 1959. Floyd's algorithm is a dynamic planning algorithm that works best for dense graphs, with positive or negative weights for paths, and is suitable for APSP (All Pairs Shortest Paths). The advantage of Floyd's algorithm is that it is simple to implement and, due to the compact structure of the triple loop, is more efficient than Dijkstra's algorithm for dense graphs. However, for searching single-source shortest paths, Floyd's algorithm will have a large amount of redundancy and will find a large number of shortest paths to unwanted nodes and take a lot of time.
Algorithm steps:

step1. Initialize a path matrix, adding the weights of the paths to the matrix if there is a path between two points, otherwise, set to infinity. Initialize a matrix of intermediate points, initializing each point's intermediate point to itself first.

step2. Loop for all nodes as intermediate nodes and update the value of the path matrix and the value of the intermediate point matrix if there is a node v that makes the path from node x to node y shorter.

step3. After three loops the algorithm ends and the desired value is obtained.

Compared to other algorithms, Floyd's algorithm has a space complexity of $O(n^2)$ and a time complexity of $O(n^3)$ compared to Dijkstra's algorithm because it relies on the path matrix and the intermediate point matrix to store the node information. However, the advantage of Floyd's algorithm is that it has only five lines of core code, which makes it extremely simple to implement. In terms of searching simple graphs, Floyd's algorithm performs essentially indistinguishably from Dijkstra's algorithm, and in terms of search results, it is similar to Dijkstra's algorithm, both outperforming DFS and BFS algorithms.

3.5. A Search Algorithm*
The A* algorithm[11,12] was co-published by Peter Hart and other scholars at Stanford Research Institute in 1968 and is a commonly used algorithm for path finding and graph traversal. The A* algorithm is based on Dijkstra's algorithm, which introduces heuristic functions and predicted costs, and is the most efficient direct search method for solving shortest paths in a static road network, as well as being a common heuristic for many other problems.

The core part of the algorithm lies in the design of the valuation function, $f(n) = g(n) + h(n)$, $f(n)$ is the estimated cost from the initial node via node n to the target node, $g(n)$ is the actual cost from the initial node to node n, and $h(n)$ is the estimated cost of the best path from node n to the target node.
Algorithm steps:

step1. Add the starting node to the priority queue.

step2. Traverse the current priority queue to find the node with the smallest F-value and make it the current node.

step3. Mark it as visited and process it for the adjacent nodes.

step4. If the neighboring node has not been visited, add it to the queue and set the current node as the parent of the newly added node, recording the F, H, and G values of the new node. If the neighboring node has already been visited, check if there is a better path. Check if the current node has a smaller G-value. If there is no smaller G value, no action is taken. If the G value is smaller, set the parent node of that node to the current node and re-set the G and H values of that node.

step5. When it happens that the target node is marked, and the path has been found. Or the priority queue is empty, terminate the algorithm. Otherwise, jump back to step 2.

step6. When the path is found, move from the endpoint to the start node along with the parent node. The A* algorithm is an advanced version of Dijkstra's algorithm compared to other algorithms. In terms of time complexity, the A* algorithm outperforms Dijkstra's algorithm and Floyd's algorithm, and can be asymptotically advanced to the $O(n)$ case.

4. Methodology; the Integrated EV–VIKOR

Figure 1 explains procedure of the integrated EV–VIKOR as follows:

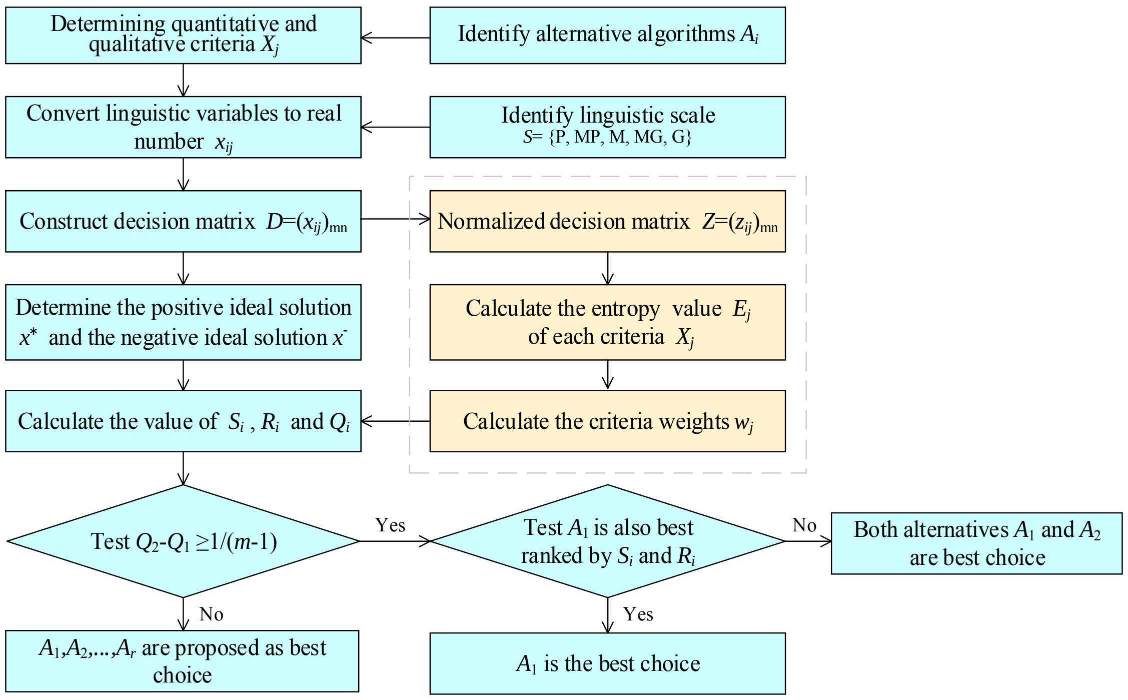
Determining quantitative and qualitative criteria X_j
Identify alternative algorithms A_i
Convert linguistic variables to real number x_{ij}
Identify linguistic scale S= {P, MP, M, MG, G}
Construct decision matrix $D=(x_{ij})_{mn}$
Normalized decision matrix $Z=(z_{ij})_{mn}$
Determine the positive ideal solution x^* and the negative ideal solution x^-
Calculate the entropy value E_j of each criteria X_j
Calculate the value of S_i , R_i and Q_i
Calculate the criteria weights w_j
Test Q_2-Q_1 ≥1/(m-1)
Yes
Test A_1 is also best ranked by S_i and R_i
No
Both alternatives A_1 and A_2 are best choice
No
$A_1,A_2,\ldots,A_r$ are proposed as best choice
Yes
A_1 is the best choice

Figure 1. Procedure of the integrated EV–VIKOR.

Consider a multi-attribute decision problem, let the set of options be $A_i(i = 1,2,\cdots,m)$, the set of attributes be $X_j(j = 1,2,\cdots,n)$, and the values of the attributes given for each objective for multiple data types be x_{ij}, assuming that the attribute weight information $w_j(j = 1,2,\cdots,n)$ is known and $w_j \geq 0, j = 1,2,\cdots,n$, $\sum_{j=1}^{n} w_j = 1$, that the weights of each criteria are derived by the Entropy Value method (EV)[13,14]. The steps of the EV-VIKOR method are as follows:

step 1.Constructing multi-attribute decision matrices $D = \left(x_{ij}\right)_{m\times n}$;

The decision-maker gives a value for each option under each attribute, both quantitative and qualitative, where the qualitative attribute can be given a linguistic value by the decision-maker based on his or her knowledge and experience.

step 2. Normalization of objective attribute values:

Before normalization, the linguistic variable data types will be transformed according to the linguistic evaluation set S= {P, MP, M, MG, G}. Here, for the quantification of qualitative attributes represented by linguistic variable data, the rating values are given on a scale of 1 to 5, with 1 being the lowest and 5 being the highest.

Let the normalized decision matrix be $Z = \left(z_{ij}\right)_{m\times n}$. The common attributes are benefit-based and cost-based indicators, such that $I_j (j = 1,2)$ denotes the set of subscripts for benefit-based and cost-based, $= \{1,2,\cdots,m\}$, For real data, the normalized data are, after normalization,

Benefit-based indicators:

$$z_{ij} = x_{ij}/max\left(x_{ij}\right) \quad ,i \in M; j \in I_1 \tag{1}$$

Cost-based indicators:

$$z_{ij} = min\left(x_{ij}\right)/x_{ij} \quad ,i \in M; j \in I_2 \tag{2}$$

step 3. Calculate the criteria weights w_j, using the Entropy Value (EV) method;

The entropy value of the real numbers under the set of criteria $X_j (j = 1,2,\cdots,n)$ is that,

$$E\left(X_j\right) = -\frac{1}{\ln m}\sum_{i=1}^{m} z_{ij}\ln z_{ij} \quad ,j = 1,2,\cdots,n \tag{3}$$

The weights w_j for each criteria $X_j (j = 1,2,\cdots,n)$ are,

$$w_j = -\frac{1-E\left(X_j\right)}{n-\sum_{j=1}^{n} E\left(X_j\right)} \quad ,j = 1,2,\cdots,n \tag{4}$$

step 4. Determine the positive ideal solution x_j^* and the negative ideal solution x_j^-, based on the decision information $D = \left(x_{ij}\right)_{m\times n}$;

There are different types of attributes, which are generally divided into benefit and cost types.

$$x_j^* = \left\{\left(\max_i x_{ij} \middle| j \in J_1\right), \left(\min_i x_{ij} \middle| j \in J_2\right)\right\} = (x_1^*, x_2^*, \cdots, x_n^*) \tag{5}$$

$$x_j^- = \left\{\left(\min_i x_{ij} \middle| j \in J_1\right), \left(\max_i x_{ij} \middle| j \in J_2\right)\right\} = (x_1^-, x_2^-, \cdots, x_n^-) \tag{6}$$

In the equation, x_{ij} is the attribute value of option A_i concerning attribute X_j, J_1 and J_2 are the sets of subscripts for the benefit-based and cost-based objectives respectively.

step 5. Calculation of group benefits of options S_i, individual regret values R_i and trade-off values Q_i.

$$S_i = \sum_{j=1}^{n} \frac{w_j d\left(x_j^*, r_{ij}\right)}{d\left(x_j^*, x_j^-\right)} \quad ,i = 1,2,\cdots,m \tag{7}$$

$$R_i = \max_j \left(\frac{w_j d\left(x_j^*, r_{ij}\right)}{d\left(x_j^*, x_j^-\right)}\right) \quad ,i = 1,2,\cdots,m \tag{8}$$

$$Q_i = v\frac{S_i - S^*}{S^- - S^*} + (1-v)\frac{R_i - R^*}{R^- - R^*} \quad ,i = 1,2,\cdots,m \tag{9}$$

In the formula, $S^* = \min_i S_i$, $S^- = \max_i S_i$, $R^* = \min_i R_i$, $R^- = \max_i R_i$, the smaller the value of S_i , Q_i and R_i, the better. v is the decision mechanism coefficient, and $v = [0,1]$, indicates the decision-maker's preference for group opinion and individual regret values. A value of $v > 0.5$ indicates that the decision-maker prefers to make decisions based on maximizing the weight of group benefits, a value of $v < 0.5$ indicates that the decision-maker prefers to minimize individual regret values, and $v = 0.5$ indicates that there is no clear preference for the decision-maker to take a balanced compromise approach. In the VIKOR method, $v = 0.5$, the equilibrium trade-off to maximize group benefits and minimize individual regrets, is generally taken.

step 6.The values of S_i , R_i and Q_i are sorted in descending order, with smaller values representing better solutions;

step 7.Determining the compromise solution.

When the following two conditions hold, the solutions can be ranked according to the magnitude of the value of $Q_i(i = 1,2,\cdots,m)$. A smaller value means that the solution is better. Let the alternative corresponding to the smallest value of A_i be Q_i. If the smallest value Q_i satisfies both of the following two conditions C_1 and C_2, then solution is A_i the best compromise solution.

Condition C_1: acceptability advantage $Q_2 - Q_1 \geq 1/(m-1)$, where Q_1 and Q_2 are the smallest and second smallest combined values ranked by $Q_i(i = 1,2,\cdots,m)$, respectively, and m is the number of options.

Condition C_2: The best option A_i also has the smallest value of S_i or R_i in order Q_i.

If the above two conditions do not hold simultaneously, compromise solutions are obtained, divided into two types: if the condition C_1 is not satisfied, then $A_1, A_2, \cdots, A_r$ are a compromise solutions, where the maximum value r of is determined by $Q_r - Q_1 \geq 1/(m-1)$. These solutions are close to the ideal solution; if the condition C_2 is not satisfied, then the alternatives A_1 and A_2 are compromise solutions.

5. Case Studies

Figure 2 below shows a road map of part of Leeds city, with the upper end of the path as the starting point A and the lower end as the endpoint Z.

Five path planning algorithms are proposed, using $A_i(i = 1,2,\cdots,5)$, which are the DFS, BFS, Dijkstra, Floyd and A* search algorithms. There are four evaluation metrics considered, using $X_j(j = 1,2,3,4)$, which are time complexity X_1, space complexity X_2, search efficiency X_3 and adaptability X_4. Metrics X_1 and X_2 are given evaluation values based on time consumption and memory size, metrics X_3 and X_4 are qualitative attributes and are given linguistic evaluation values by the decision-maker, metrics X_1 and X_2 are cost-based metrics, metrics X_3 and X_4 are benefit-based metrics, and the path planning algorithm decision information matrix is shown in Table 1 below.

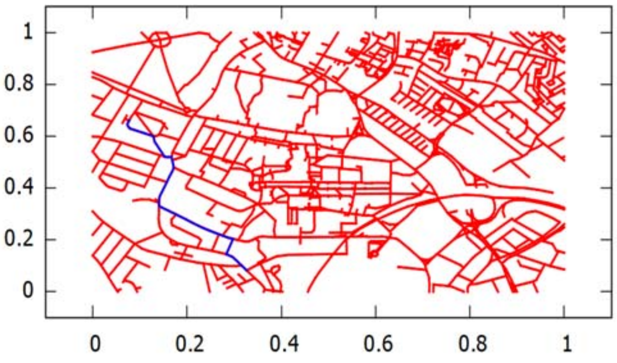

Figure 2. Map of part of Leeds city roads.

Table 1. Formatting sections, subsections and subsubsections.

	X_1	X_2	X_3	X_4
A_1	$O(n^2)$	$O(n^2)$	low	poor
A_2	$O(n^2)$	$O(n^2)$	medium low	medium poor
A_3	$O(n^2)$	$O(n^3)$	medium high	medium
A_4	$O(n^3)$	$O(n^2)$	medium	medium good
A_5	$O(n)$	$O(n^2)$	high	good

The initial decision matrix is mixed information, where all data is converted to a uniform real data type due to the inconvenience of calculation.
The indicator attribute values are quantified according to the algorithm consumption time according to a scale of 1 to 9 levels, where 1 corresponds to the time complexity $O(1)$ and 9 corresponds to $O(n^n)$; the indicator attribute value quantization processing is related to the temporary memory space occupation of the coding implementation, quantified by the actual value and processed regarding a scale of 1 to 9 levels, while, at the same time, allowing the quantified value of the attribute to maintain some variability; the indicator X_3 and X_4 are quantified according to the language evaluation sets are quantified according to a scale of 1 to 5, with poor (low) corresponding to 1 and high (good) corresponding to 5.
Normalized attribute matrix $\left(z_{ij}\right)_{5\times4}$, as shown in Table 2 below.

Table 2. Normalized attribute matrix.

	X_1	X_2	X_3	X_4
A_1	0.2083	0.1939	0.0667	0.0667
A_2	0.2083	0.1939	0.1333	0.1333
A_3	0.2083	0.2091	0.2667	0.2000
A_4	0.2500	0.2015	0.2000	0.2667
A_5	0.1250	0.2015	0.3333	0.3333

Based on the normalized decision matrix, the weights of each attribute calculated according to the entropy method are $w_j(j = 1,2,3,4)$:

$$w = (0.0860, 0.0015, 0.4563, 0.4563)$$

The positive ideal solution x_j^* and the negative ideal solution x_j^- are:

$$x_j^* = (3,5.1,5,5)\ ,\ x_j^- = (6,5.3,1,1)$$

The group utility values S_i, individual regret values R_i and trade-off values Q_i for each algorithm , and are sorted in descending order as shown in Table 3 below, where is taken $v = 0.5$.

Table 3. Group utility values S_i, individual regret values R_i and trade-off values Q_i.

	S_i		R_i		Q_i	
	Calculated values	Sorting	Calculated values	Sorting	Calculated values	Sorting
A_1	0.4213	1	0.2082	1	1.0000	1
A_2	0.3172	2	0.1561	2	0.7515	2
A_3	0.1611	4	0.1041	3	0.4412	4
A_4	0.1635	3	0.1041	3	0.4441	3
A_5	0	5	0	4	0	5

From the calculated values Q_i, the path planning algorithm is ranked as $A_5 \succ A_3 \succ A_4 \succ A_2 \succ A_1$, where the algorithm corresponding to the smallest value of Q_i is A_5, and then check whether the algorithm A_5 satisfies the other two conditions C_1 and C_2. For the condition C_1, $Q(A_3) - Q(A_5) = 0.4412 > 0.25$, the condition C_1 is satisfied; for the condition C_2, the value S_i of for algorithm A_4 is the smallest and the condition C_2 is satisfied. Therefore, the algorithm A_5 is the best algorithm for path planning for urban vehicles in Leeds.

6. Conclusions and Further Discussion

This paper addresses the arbitrariness and unreasonableness of the current practice of choosing path planning algorithms based on experience alone, and attempts to propose a decision analysis method for choosing path planning algorithms, which provides a scientific decision-making tool for people to choose the appropriate algorithm for path planning.

Although the case study in this paper is focused on the selection of path planning algorithms for urban vehicles in Leeds, evaluation metrics can be added or subtracted as needed to apply to the selection of path planning algorithms in different application areas and complex environments.
There are also some shortcomings in the research presented in this paper. The selection of path planning algorithms is a multi-attribute decision problem, which requires a comprehensive analysis of various evaluation factors and a complete and comprehensive evaluation index system. influencing factors are not comprehensive enough, which is the shortcoming of this paper and a topic worthy of further research in the future.

References

[1] Li, Y.D.; Ma, T.L.; Chen, C.B.; Wei, H.L.; Yang, Q.G. A review of path planning algorithms for unmanned vehicles. *Foreign Electronic Measurement Technology* 2019,38, pp. 72-79.
[2] Huo, F.C.; Chi, J.; Huang, Z.J.; Ren, L.; Sun, Q.J.; Chen, J.L. A survey of path planning algorithms for mobile robots. *Journal of Jilin University (Information Science Edition)* 2018,36, pp. 639-647.
[3] Gu Lei. Vehicle routing algorithm and its application review. *Logistics Engineering and Management* 2019,41, 100-101+33.
[4] Zhao, W.J.; Gong, Z.Y.; Wang, W.; Fan, S.F. Comparison and analysis of several classical shortest path algorithms. *Journal of Chifeng University (Natural Science Edition)* 2018,34, pp. 47-49.
[5] Liang, X.H.; Mu, Y.H.; Wu, B.H.; Jiang, Y. A review of correlation algorithms for path planning. *Value Engineering* 2020, 39, pp. 295-299.
[6] Chen, Q.J.; Jin, Y.Q.; Han, L. Review of UAV path planning algorithm. *Flying Missiles* 2020,pp. 54-58.
[7] Yang, X.; Wang, R.; Zhang, T. Review of intelligent optimization algorithms for path planning of UAV cluster. *Control Theory & Applications* 2020,37, pp. 2291-2302.
[8] Zhang,K.Review of path planning algorithms for unmanned vehicles. *Equipment Manufacturing Technology* 2021,6, pp.111-113.
[9] Dijkstra, E.W. A note on two problems in connexion with graphs. *Numerische mathematic* 1959,1, pp. 269-271.
[10] Floyd, R.W. Shortest path. *Communications of the Association for Computing Machinery* 1962,15, pp. 245.
[11] Hart, P., Nilsson, N., Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 1968,4, pp. 100-1
[12] Dieter, F. A probabilistic approach to collaborative robot localization. *Autonomous Robots* 2008,8,pp. 325-344.
[13] Opricovic, S. Multi-criteria optimization of civil engineering systems. *Faculty of Civil Engineering*, Belgrade 1998,2, pp. 5-21.
[14] Zhu, Y.; Tian, D.; Yan, F. Effectiveness of entropy weight method in decision-making. *Mathematical Problems in Engineering* 2020, 3564835.