ATLANTIS PRESS

Advances in Computer Science Research, volume 87

3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)

Research on Optimal Path based on Dijkstra Algorithms

Minhang Zhou [a], Nina Gao [b]

Chengdu University of Information Technology Chengdu, China

[a] 435446921@qq.com, [b] galing2003@cuit.edu.cn

Abstract. The optimal path selection is a practical problem, and the application frequency in daily life is very high, which has practical research significance. The Dijkstra algorithm is one of the most classical algorithms in the algorithm of optimal path selection. It is also the theoretical basis of many experimental designs when solving the optimal path problem. When the traditional Dijkstra algorithm calculates and selects the optimal path between nodes, a large number of nodes need to be calculated. The selection of the optimal path will be affected by many conditions. In the actual application, the selection conditions will often present multiple different priority conditions and comprehensive influence path selection. In this paper, the optimization algorithm is implemented and verified by experiments and practical applications.

Keywords: optimal path, Dijkstra algorithm, optimization.

1. Overview

Since the advent of the computer, people have paid more and more attention to the optimal path selection problem, and at the same time, the update and optimization of the algorithm has been continuously improved. At that time, scientists made continuous attempts to implement the optimal path selection program. The representative one was proposed by Dijkstra: the main problem solved by the algorithm in the optimal path selection process is the optimal path problem of two points. However, it was not until 1959 that Edsger Wybe Dijkstra proposed the basic idea of solving the optimal path selection problem from the origin to other nodes and gave the original algorithm. This is the algorithm later called Dijkstra, and the Dijkstra algorithm has also become a milestone classic.

2. Overview of Optimal Path Algorithms at Home and Abroad

At present, the research on the optimal path mainly includes two aspects. The first aspect is the optimal path problem: Bellman, Dijkstra, and Dreyfus have studied and designed many effective algorithms when studying this optimal path problem. Some of the algorithms developed have become the optimal algorithm algorithms under certain circumstances. However, when the conditions are not certain, the selection of the optimal path includes other aspects. Among them, Frank and Mirchandani have studied the random variation of the length of the link, and the condition is not the most independent of time. Excellent path problem; Loui, Muethy and Sarkar consider the conditions under which the cost function affects the optimal path selection when the urban road network design is established. Hall (1986), Li Ping Fu, and LRRilett (1998), Elise and Hani (2000) studied the optimal path problem for a combination of two conditions with randomly varying road length and time independence. Tomas and Rajeev studied the optimal path selection problem with link length spacing. Through the research and optimization of the optimal path problem by various scientists, the selection of the optimal path is now a mature and perfect problem. The mainstream algorithms are as follows.

The A* algorithm solves and selects the optimal path effective is a method used in static road networks. The basic formula of the A* algorithm is expressed as:

$$f(n)=g(n)+h(n) \tag{1}$$

The A* algorithm is a heuristic search algorithm in artificial intelligence algorithms. The optimization of the A* algorithm is that the consistent road network information and the information of the known target nodes are first introduced. When the nodes that have been explored are selected, the node information is directly called. The function is then used to evaluate the separation distance

between the selected node and the target point as a condition for selecting the node to pass through the next path. The advantage of the A* algorithm is that it does not need to traverse all nodes, but instead proceeds in the direction of the desired road (the target node that needs to be experienced) according to the selected heuristic function. In the actual application process, the A* algorithm often finds the optimal path needed because the heuristic function is not suitable. Therefore, the success rate of using the A* algorithm to select the optimal path is not very high.

Genetic Algorithms (GA) is a computational model that simulates the evolutionary processes of organisms such as genetic selection and natural elimination in the natural environment. He is inspired by Darwin's theory of evolution. Since the genetic algorithm is based on the inheritance of genetics and the survival of the fittest in the inferior, the iterative process of the search algorithm itself also has a special process for detecting and retaining the data to be processed.

In the calculation process of the genetic algorithm, the solution of each possible case is designed into a single vector, which is called a "chromosome" in the genetic algorithm. The individual element in each of the vectors is called a "gene". All the vectors (chromosomes) in the algorithm form the whole, and each vector (chromosome) is evaluated and screened according to the designed objective function in the calculation process. According to the final result of the algorithm, a representation is expressed as “fitness”. value. When the genetic algorithm starts to calculate, a part of the information "chromosome" is randomly generated, and the "fitness" of the randomly generated chromosome is calculated, and in the subsequent algorithm, the "chromosomes" are based on different "fitness" (Conditional information) Performing operations such as “selection, exchange, mutation”, etc., gradually eliminates chromosomes with low fitness in the calculation process, that is, excludes paths with low priority in the path selection process, leaving a “chromosome” with high fitness, ie Preserve the optimal path that meets the criteria.

Because the "new generation" screened by the algorithm is selected from the "old generation" conditions, the new data obtained is more eligible data from the overall trend, that is, better than the previous generation data. The genetic algorithm is iteratively iterated in the calculation process to know that the final optimization results that meet the preset conditions are finally obtained.

3. Dijkstra Algorithm

The basic idea in the Dijkstra algorithm is roughly as follows: First, set a set S of vertices, and the path weights from the source point (starting point) S to the vertices in the set have been determined. During the operation of Dijkstra algorithm, different paths are repeatedly selected, that is, the vertices with the optimal path estimation are added, and the vertex i is added to S, and all the path weights of the vertex i are weighted by the function, and the final function weight value is selected. The smallest path is the optimal path.

4. Dijkstra Algorithm Principle

The Dijkstra algorithm can select the optimal path that meets the conditions from the topology roadmap. The traditional Dijkstra algorithm divides the nodes in the topological network into three parts: first, all the initial nodes in the algorithm network are unmarked nodes, and the nodes that need to pass and connect in the process of path selection and optimal path selection For the temporary node, each screening cycle in the optimal path selection process selects the node with the shortest path length from the temporary marker node as the permanent marker node, and the Dijkstra algorithm will continue until the target node or all the nodes. The node will not end until it becomes a permanent marker node.

In the process of selecting the optimal path, each node is given a pair of labels, marked as (,), which represents the obstruction weight of the shortest path from the origin point s to the point j; the representation is from s The barrier weight of the shortest path of the j-node from the (origin node) to the shortest path of j (the target node). The process of selecting the optimal path using the Dijkstra algorithm is as follows:

In the first step, we need to mark each node that is waiting to pass through: set the starting point of the path, that is, the origin point to S point, and the length from the origin point to its own is 0; other nodes that need to pass through are set to w_s=∞, p_j; mark the origin point s, all other nodes are set to untagged.

In the second step, the distance of all the marked points K to the unmarked points J directly connected to the marked points is calculated, and the function is set to:

$$w_j=\min \{ w_j,w_k+d_{kj} \} \quad (2)$$

In the function expression, it is the direct connection distance from point k to j.

The third step is to select the next point. Among the unmarked nodes, select the smallest one i (usually i=1): another w_i=min w_j, where j is the order of all unmarked points, Point i will be selected as the point required in the optimal path selection process, and the node i identification will be changed to the marked point.

The fourth step is to find the previous point of node i. Find the point j^* that is directly connected to point i from the marked points. As the previous point, set: $i=j^*$.

In the fifth step, node i is marked. If all nodes are marked, the algorithm is fully launched, otherwlise, remember k=i, go to step (2) and continue with the steps.

5. Advantages of the Algorithm

The Dijkstra algorithm can find all the optimal paths and the correct rate of these optimal paths is 100%, but the search speed is slow, and the time-consuming efficiency is low in multi-node screening. Generally, it can only be used for non-online path selection and planning or nodes. Or the optimal path planning problem with less conditional information. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

For a graph with n nodes, in the Dijkstra algorithm, it is necessary to perform n-1 iterations on the road map with n nodes. Each iteration adds a node as a marker node to the corresponding node set, and experiences the i-th. After the iteration, it is necessary to process the information obtained by the ni nodes, and block the weight and other information. Therefore, the information of the n node networks is processed. The time complexity is O(n^2).

6. Comparison of Dijkstra Algorithm with other Mainstream Algorithms

Cherkassky et al. selected 17 better representative algorithms in the optimal path algorithm at the end of the 20th century. The result is that no algorithm can be applied to all road network models. In the 1990s, Zhan and Noon applied the Cherkassky test algorithm to the actual traffic network construction process. The test results show that the Dijkstra algorithm has advantages in the calculation and screening of the optimal path between two points.

When the number of nodes that need to pass through the route topology map is relatively small, the time spent by Dijkstra algorithm, A* algorithm and genetic algorithm is almost the same, when the number of nodes that need to be traversed is required. After the increase, the A* algorithm has the fastest screening speed, the genetic algorithm is second, the Dijkstra algorithm is the slowest, and the gap and change between the three will become more significant as the number of nodes increases. That is, when the number of nodes to be calculated and the number of roads are large, the Dijkstra algorithm is slower than the other two algorithms.

When the number of nodes that need to pass through the route topology map is relatively large, it can be seen that the Dijkstra algorithm's path selection accuracy and the number of selected paths are 100%, and the A* algorithm is the lowest. The success rate of the search for the optimal path is higher

than that of the genetic algorithm. This is because the Dijkstra algorithm is a traversal algorithm that guarantees 100% search for the optimal path each time.

7. Dijkstra Optimization Algorithm Research

The Dijkstra algorithm is used to solve the optimal path from the node (source point) in the route topology to the remaining nodes. The time complexity is O(N2); after the condition is selected, the selected condition is formed into a new matrix as the optimal path weight matrix as the path selection condition. For example, in the process of logistics and distribution, it is necessary to go through a number of necessary locations, and the source points are known. However, in the process of selecting the path, it is necessary to pass through a plurality of necessary nodes. The selection of these nodes has nothing to do with other factors. On this basis, it takes the least time, or the travel is the safest because of the particularity of the item, and this new weight is formed. The matrix selects the route. Therefore, the choice of the optimal route will be more in line with the needs of the user.

In this design, the optimization algorithm was demonstrated with the logistics distribution as the scenario. In the process of logistics and distribution, the merchant needs to reach all the customer addresses in the shortest time. The situation is different. There are other requirements for the selection of the route. For example, the safety factor of the route selection is required in the process of distributing large valuables. There is a high requirement, and there must be a point location that must arrive in the path selection process. Therefore, in the implementation process of the program, each node appearing in the topology map needs to be traversed, thereby obtaining an optimal path that meets the condition. . In this paper, four conditions are selected as the requirements for path selection. Four conditional matrices are formed respectively: time matrix T, path matrix S, safety factor matrix A and comfort matrix H for the distributor. In the path selection process, four inputs are required first. The requirement level of the condition is usually 1-3, and then the four conditional matrices are combined and calculated by the function calculation tool numpy in Python to form the weight matrix K used in the Dijkstra algorithm.

The specific formula is:

$$K = \sum_{ij=0}^{n} (aX_{ijs}T_{ij} + bX_{ijs}S_{ij} + cX_{ijs}A_{ij} + dX_{ijs}H_{ij}) \tag{3}$$

From this, we have obtained a new weight matrix that can be used in the calculation of Dijkstra algorithm, and finally realize the selection of the optimal path in Pycharm using the new weight matrix.

8. Optimization Algorithm Description

In the road selection process, vehicles from vehicles such as vehicles are used to reach the destination Dj event and select the time, path length, comfort, and security level of the route selection, and count into the optimal path selection process. . The time matrix A, the path matrix B, the comfort matrix C, the security matrix D. Divide the grade into 1, 2, 3, set x, y, z, k, and assign the grade value, select the appropriate coefficient and multiply it by the matching matrix to get the four condition variable time matrix, the path matrix, comfortable. Degree matrix, security matrix A1, B1, C1, D1. The matrix I, J, Q, P four matrix summation (Equation 3-1) forms a weight matrix Z for Dijkstra matrix optimal path selection.

The Dijkstra algorithm solution process:

The first step is to identify the node set as S = {s}; the unidentified node set is T, and the number of set T is the summary point number minus the identified node set S, that is, MS; w_j is the optimal path length of j from the source point s to the node, w_j = d_ij when node s and node j can be directly connected; w_j = ∞ when node s and node j are not directly connected.

The second step: find a new node i in the unidentified node set T, so that the weight between the origin node s and the target node i is the smallest, and then add the target node i to the marked node

set S. That is, when d_si = min d_sj,, the set of nodes has been identified as S = {s, i}, and the set of unidentified nodes is T = T-{i}; j∈T.

The third step: modify the value of a single node in the unidentified node set T, here we assume node j as the target node, which can be expressed as W_j = min(w_j, w_i + d_ij); where when the value of the target node w_j changes, the node p_j = i.

The fourth step: Select the minimum obstruction weight of all nodes w_j, and classify the filtered nodes into the identified node set S, which can be expressed as:

W_i = min w_i; S=S∪{i}; T=T-{i};

when the number of nodes in the identified node set S is N, all nodes have completed the identification, and the algorithm ends. Otherwise, the calculation step needs to be returned to the second one for loop calculation until the number of nodes is N.

9. Characteristics of the Optimized Algorithms

The traditional Dijkstra algorithm is based on the breadth-first search strategy. Starting from the specified node, iteratively iterating through all other nodes by weights, and finally obtaining the optimal path tree from the specified node to other nodes. It uses a linear array structure to store its association matrix. It needs to access all untagged nodes when extracting the optimal path node. The running time of the algorithm is O(n2). The Dijkstra algorithm is used to solve the optimal path from any node (source point) to the rest of the nodes on the graph. After selecting conditions for the desired path, the selected conditions form a new matrix as the optimal path weight matrix as the path selection condition. The scenario selected in this paper is that it needs to go through multiple necessary locations in the logistics distribution process. In the process of selecting the path, it is necessary to pass through a plurality of necessary nodes. The selection of these nodes has nothing to do with other factors. On this basis, it takes the least time, or because the speciality of the item requires the safest route, the newly formed weight matrix pair Route selection. Therefore, the choice of the optimal route will be more in line with the needs of the user.

10. Implementation of Optimized Dijkstra Algorithms

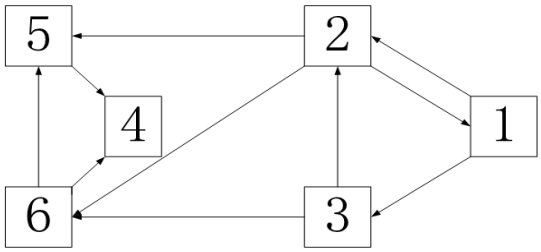

Figure 1. Path topology graph

```
def get_min():
    if (debug):
        print("[get_min]")
    if (len(dist) < 1):
        return {'index': -1, 'val': 0}
    i = 0
    min_val = MAX_NUM
    min_index = 0
    while (i < len(dist)):
        if (v_in_S(i + 1) == "NO"):
            if (dist[i] < MAX_NUM and dist[i] > 0 and min_val > dist[i]):
                min_val = dist[i]
                min_index = i
        i = i + 1
    return {'index': min_index + 1, 'val': min_val}
```

Figure 2. Function get_ min ()

Figure 1 shows one of the route topologies that needs to be used in the real-time process of the program. There are 6 nodes in the experiment. It is assumed that node 1 is the origin point, and node 2 to 5 need to be traversed.

The origin point initial flag is set to 1, and the path distance from the origin point to the own node is set to 0, while the other nodes are tagged.

All six nodes that need to go through the experiment are marked as 1 to 6, and these nodes are placed in the path node set S.

By using the function get_min in Figure 2, we can compare the dist values of the node with the next connected node after each node is experienced, and the next node with a smaller dist value is selected as the selected path.The value of i is incremented by one to start. Every time you go through a node, you need to repeat this step to know that all nodes have been completed.

The dist value is updated after each node experience, and the new dist value is used when the next node is selected. The dist value is updated after each node performs the minimum comparison and the next node selection during the running of the program. During the running of the program, the dist value is updated cyclically, and the final result is completed.

It is used to combine the functions used by Dijkstra algorithm in the process of program realization, and finally realizes the selection of the optimal path. It runs in pycharm and outputs the correct optimal path selection result.

```
初始条件:
    起始位置= 1
    起始点= [{'index': 1, 'val': 0}]
    dist 值= [0, 30, 15, 10000, 10000, 10000]
结果:
 节点信息= [{'index': 1, 'val': 0}, {'index': 3, 'val': 15}, {'index': 2, 'val': 25},
{'index': 6, 'val': 30}, {'index': 5, 'val': 40}, {'index': 4, 'val': 50}]
    dist 值= [0, 25, 15, 50, 40, 30]
```

Figure 3. Choosing the optimal path with time as a single condition

When a single time is the optimal path selection condition, all six nodes are traversed, and the optimal path given by the program according to the priority time of each path is 1-3-2-6-5-4, the values of the dist weighting function are 0, 25, 15, 50, 40, 30, respectively. The total time that the selected time optimal path suffers is 160. If 1 minute is used as the weight 1 of the weight, the total time used for the path is 160 minutes, as is shown in Figure 3.

```
初始条件:
    起始位置= 1
    起始点= [{'index': 1, 'val': 0}]
    dist 值= [0, 20, 25, 10000, 10000, 10000]
结果:
 节点信息= [{'index': 1, 'val': 0}, {'index': 2, 'val': 20}, {'index': 3, 'val': 25},
{'index': 5, 'val': 30}, {'index': 6, 'val': 35}, {'index': 4, 'val': 45}]
    dist 值= [0, 20, 25, 45, 30, 35]
```

Figure 4. Choosing the optimal path with comfortability as a single condition

When using the route as the selection condition for selecting the optimal path, when we want to go through the other 5 locations starting from the starting point 1, we can see from the program realization chart that the best path selected is 1-2-3. -6-5-4, dist weighting function values are 0, 15,

30, 65, 45, 35 respectively. Assuming that 1 km is calculated as the weight 1 and the total stroke of the path is 190 km. It can be seen that there is not much difference compared with the time. The relationship between the driving distance and the consumption time is relatively tight. There is no other influence at this time, and the selected path appears the same, as is shown in Figure 4.

When the comfort level of one path is taken as the single condition variable when the optimal path is selected, the path taken out and the path condition are selected, eventually the path selected by the time condition is very different.

In the process of selecting this route, we have to go through the other 5 locations from the starting point 1. From the program realization chart, we can see that the best path selected is 1-2-3-5-6-4. The dist weighting function values of the best path are: 0, 20, 25, 45, 30, 35. Here we can assume the smoothness of the road surface, rest the building along the way or the scenery along the way as a factor of influence on the comfort, the lower the weight, the higher the comfort, from Figure 4 we can see the optimal path of the selected The total dist weighting function value is 155.

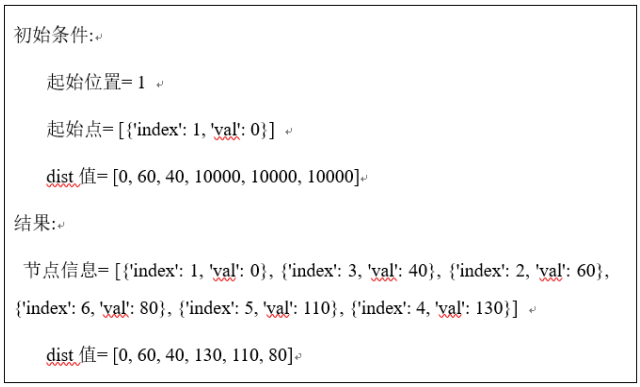

Figure 5. Choosing the optimal path with safety degree as a single condition

When the comfortability of the road is selected as the single condition of the optimal path selection, the optimal path and distance are compared, and the time optimal path changes greatly. In real life, the comfort level of travel when traveling is one of the main factors that people pay attention to. The priority will be higher than the distance and time factor under certain circumstances; but because comfort is the subjective influence factor, for different people Classification will also have different requirements, so the controllability is low, and it is necessary to continue the representationalization, split the comfort into representative small factors or collect and analyze the user's subjective needs in the form of questionnaires before the path selection. The adjustment of the comfort matrix weight can be performed as a follow-up to the algorithm optimization.

When the safety degree of the path is taken as the single variable for selecting the optimal path, the selected optimal path will be different from the comfort.

In the process of selecting this route, we have to go through the other 5 locations from the starting point 1. From the program realization chart, we can see that the best path selected is 1-3-2-6-5-4. The dist weighting function values of the best path are 0, 60, 40, 130, 110, 80. Here we can assume whether the road is in the accident-prone area, whether there will be steep slopes in the path, sharp bends and other factors as factors influencing the safety factor when selecting, the lower the weight, the higher the safety factor, we can see from the figure The selected optimal path total dist weighting function value 420 is selected.

When the safety factor is selected as the single selection condition, the optimal path is different from the comfort path as the single condition. In the actual path selection process, the safety factor of each path will change with time, environment and other factors. For example, the safety factor of the rainy mountain road will decrease and the weight will increase. The safety factor of the road without the street lamp will also decrease. Obstructing the increase of the weight, the selection of the safety factor will be affected by the change of external factors. In this experiment, the rainy day variable option can be added. When the user selects a rainy day, it will also affect the original matrix of the safety factor, and the weight of the special path will change and have an impact. In the actual situation,

there are other factors that affect the safety factor, which can be used as an extension of algorithm optimization.

```
>>> x=int(input('please input 路程优先级：'))
please input 路程优先级：1
>>> y=int(input('please input 时间优先级：'))
please input 时间优先级：2
>>> z=int(input('please input 舒适度优先级：'))
please input 舒适度优先级：3
>>> a=int(input('please input 安全优先级：'))
please input 安全优先级：4
>>> from numpy import *;
>>> 等级调整矩阵1=mat(ones((6,6))*x)
>>> 等级调整矩阵2=mat(ones((6,6))*y)
>>> 等级调整矩阵3=mat(ones((6,6))*z)
>>> 等级调整矩阵4=mat(ones((6,6))*a)
>>> 路程矩阵=mat(random.randint(20,size=(6,6)))
>>> 时间矩阵=mat(random.randint(20,size=(6,6)))
>>> 安全矩阵=mat(random.randint(20,size=(6,6)))
>>> 舒适度矩阵=mat(random.randint(20,size=(6,6))
>>> 路程1=multiply(路程矩阵,等级调整矩阵1)
>>> 时间1=multiply(时间矩阵,等级调整矩阵2)
>>> 舒适度1=multiply(舒适度矩阵,等级调整矩阵3)
>>> 安全1=multiply(安全系数矩阵,等级调整矩阵4)
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    安全1=multiply(安全系数矩阵,等级调整矩阵4)
NameError: name '安全系数矩阵' is not defined
>>> 安全1=multiply(安全矩阵,等级调整矩阵4)
>>> A=路程1+时间1+安全1+舒适度1
```

Figure 6. Conditional matrix computation

By using the NumPy function tool in python, we can select the required level of the four conditional matrices, and adjust the four conditional matrices according to the corresponding levels to form the comprehensive matrix of the route weights. The generated matrix is tested on the calculation method, and the generated 6*6A matrix will be used for the optimal path filtering operation (Figure 6).

When the four conditions of distance, time, safety factor and comfort affect the selection of the optimal path at the same time, the user needs to select the priority level of the four conditions. The level is divided into 1-3 levels. When the election is completed, the program will regard the selected level as a coefficient to calculate four new conditional matrices with the corresponding matrix. When the four new matrices are added together, the weight matrix formed will be used as the final matrix A of the program running using the dijkstra algorithm.

```
初始条件:
    起始位置= 1
    起始点= [{'index': 1, 'val': 0}]
    dist 值= [0.0, 375.0, 295.0, 100000.0, 100000.0, 100000.0]
结果:
 节点信息= [{'index': 1, 'val': 0}, {'index': 3, 'val': 295.0}, {'index': 2, 'val':
375.0}, {'index': 6, 'val': 545.0}, {'index': 5, 'val': 675.0}, {'index': 4, 'val':
905.0}]
    dist 值= [0.0, 375.0, 295.0, 905.0, 675.0, 545.0]
```

Figure 7. Optimal path selection

Figure 7 shows the optimal path obtained when the conditions of time, distance, safety factor, and comfort are divided into different priorities as the path selection conditions.

11. Summary

Based on the Dijkstra algorithm, this paper optimizes the selection of multiple conditions and multiple requirements for the Dijkstra algorithm in the practical application. The optimization algorithm combines various influence elements to form a new weight matrix as a new condition for path selection by Dijkstra, which improves the search efficiency and satisfies people's various needs for path selection. The study of the Dijkstra algorithm achieves the following goals:

(1) Select various factors appearing in the path selection process, select coefficients according to different priority levels, and form a new weight matrix.

(2) Use the Dijkstra algorithm to traverse all the paths and select the optimal path that meets the conditions.

(3) Compare and analyze the new path selected under multiple conditions and the path selected by a single condition.

What is lacking in this graduation design is that the optimization of Dijkstra algorithm is relatively simple. Considering that the optimal path will have more interactive conditions in practical applications, the optimization of the program at this stage can not meet the optimal path in more complicated situations. The selection, such as a variety of conditions affect each other, some data needs to be updated in real time. In terms of programming, Dijkstra's implementation code is slightly simpler. When the condition increases, the calculation of invalid nodes will increase the running time of the program, reduce the efficiency, and the program has the ability to continue optimization.

References

[1]. Chen Yifu, Lu Wei, Ding Haojie. Research on Optimization Strategy of Dijkstra Algorithm[J]. Computer Technology and Development, 2006, 16(9): 73~75.

[2]. Yu Hen. Python3 study notes [M]. China Industrial Letter Publishing Group, Electronic Industry Press, 31-49.

[3]. Zhang Yonglong. Optimization of Dijkstra optimal path algorithm[J]. Journal of Nanchang Institute of Technology, 2006, 25(3): 30~33.

[4]. Eric Matthes (Author) Yuan Guozhong (Translator). Python Programming: From Getting Started to Practice.

[5]. [United States] Barry P. In-depth Python. Books published by Southeast University Press, 2011.

[6]. [United States] Y. Daniel Liang with Li Na (Xi'an University of Electronic Science and Technology) translation. Python language programming [M]. Mechanical Industry Press.2004.1.

[7]. A generalization of Dijkstra’s shortest path algorithm with applications to VLSI routing. Journal of Discrete Algorithms www.elsevier.com/locate/jda.

[8]. Edited by Bill Lubanovic. Python Language and Its Applications [M]. Published by People's Posts and Telecommunications Press, 2015.

[9]. A.V. Goldberg, A simple shortest path algorithm with linear average time, in: Proc. 9th European Symposium on Algorithms (ESA 2001), in: Lecture Notes in Computer Science (LNCS), vol. 2161, Springer-Verlag, 2001, pp. 230-241.

[10]. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time, in: Journal of the ACM ,1999, (46) pp.362-394.

[11]. YZ Chen. SF Shen. T Chen. R Yang. Path Optimization Study for Vehicles Evacuation Based on Dijkstra algorithm. Science Direct Procedure Engineering71(2014)159 – 165.