An Optimized Floyd Algorithm for the Shortest Path Problem

Dachuan Wei
Science and Technology Industry Division, Jilin Architectural and Civil Engineering Institute, Changchun, China
E-mail: weidachuan2010@163.com

Abstract—There is too much iteration when using the traditional Floyd algorithm to calculate the shortest path, that is to say, the computation of traditional Floyd algorithm is too large for an urban road network with a large number of nodes. As for this disadvantage, the Floyd algorithm was improved and optimized further in this study; moreover, the improved Floyd algorithm was used to solve the shortest path problem in car navigation system, and it achieved good results. Two improvements were done: firstly, construct an iterative matrix for solving the shortest path, all the nodes in it were compared at first, and those nodes which have nothing to do with the results were removed, then search for the next node directly to reduce the frequency of iteration; Secondly, construct a serial number matrix for solving the shortest path, it was used to record the case of inserting node in the process of iteration. Finally, the frequency of iteration and the computational complexity were compared for both the traditional Floyd algorithm and the improved Floyd algorithm. The experimental results and analysis showed that the computational complexity of the improved Floyd algorithm has reduced to half of that of the traditional algorithm. What’s more, by checking the iterative matrix and the serial number matrix, the shortest path can be found out simply, intuitively and efficiently.

Index Terms—improved Floyd algorithm, the shortest path, route selection, Car Navigation System

1. INTRODUCTION

With the emergence and development of computer, the research on graph theory has received wide attention, such as Dijkstra algorithm in 1950s, Floyd method in 1960s, and so on. However, research on the shortest path problem was developing slowly from 1970s to 1980s, but after entering 1990s, some new issues arose, along with the development of information science, modern communications and intelligent networks, which are the basal deformation of the shortest path problems. Therefore research on the shortest path problem began active, and now, many scholars have begun to study the issue. As a typical problem in graph theory, shortest path problem has been used in many fields. In the field of network communication, path selection problem of packet transmission is also closely related to the shortest path problem. Moreover, algorithms to solve shortest path problems have a strong value in various engineering fields. The shortest path problem is a classic problem in network optimization, and it had been solved well; the model can be used for many optimization problems, such as, equipment update, pipe laying, line arrangements, and plant layout and so on. Dynamic programming solution and graph theory solution are common methods used to solve the shortest path problem, but, construction of dynamic programming equation is difficult for some of the most shortest path problems (such as the road can not be neatly divided to sections), relatively, graph theory solution is more effective.

In modern society, with the development of the social economic, traffic jams clogging becomes more and more serious because of the popularity of cars, which not only caused inconvenience to people's travel, but also affected the rapid development of the social economic. Urban traffic congestion has become a bottleneck for constraining the development of medium and large cities at home and abroad. It not only caused serious economic losses, but also is the important reason of generating traffic accidents, environmental pollution and energy waste.

Car navigation system [1] is a kind of driving auxiliary equipment emerging in recent years. Through the commercial communication satellites, the GPS is applied to vehicle navigation for guiding the motorist, which would become a vehicle navigation system (also known as car navigation systems, can be abbreviated as CIPS). As long as the destination is inputted the car navigation system, the system will automatically calculate the most appropriate route according to the electronic map, and remind the driver to drive in accordance with the calculated routes in the process of motion. Throughout the moving process, the drivers do not have to consider which route will get them to the destination, so they will arrive quickly.

The primary task of vehicle navigation systems is to provide the best route for the drivers; therefore, the function of route selection becomes particularly important in vehicle navigation systems. The core algorithm for the function of route selection is the optimal path selection algorithm, currently, the algorithms are often used mainly including Dijkstra A*[2, 3], TC B-method and Floyd [4-6] and so on. Among many algorithms, the traditional Floyd algorithm is globally optimum, and the main program is simple, easy to program to implement, good to be coupled with other program modules and easy to be modified or added. However, this algorithm is an exhaustive algorithm, when the shortest path among each pair of vertices is computed,
all vertices with connectivity are considered as intermediate vertices to compute in the network, the time complexity is \(O(n^3)\) [7]. Therefore, the process of calculation takes a huge amount of storage (need to establish adjacency list or adjacency matrix) and a huge amount of costs of calculation (with the third power of the network nodes), so only suitable for resolving the dense graph with small scope and less nodes [8].

However, for the urban road network with lots of nodes, it is necessary to reduce the computational complexity of the shortest path for route selection capability of car navigation system being able to quickly and accurately choose the best path for the driver. Therefore, in this paper, traditional Floyd algorithm has been improved by constructing a iterative matrix and serial number matrix, and the improved Floyd algorithm is applied to vehicle navigation system to implement the route options.

II. TRADITIONAL FLOYD ALGORITHM

A. Related Definitions

Definition 1: A directed graph \(D\) is a binary group constituted of a non-empty finite set \(V(D)\) and a ordered pair set \(A(D)\) made up of certain elements in \(V(D)\), it can be denoted by \(D=(V(D), A(D))\), abbreviated as \(D=\langle V, A \rangle\). Where, finite set \(V\)={\(v_1, v_2, \ldots, v_n\)} is the vertex set of \(D\); \(A\)={\(a_{11}, a_{12}, \ldots, a_{nn}\)} is the arc set of \(D\), and each element \(a_{ij}\) (1\(\leq\)\(i\), \(j\)\(\leq\)m) in \(A\) is an ordered pair of two elements \(v_i\) and \(v_j\) (1\(\leq\)\(i\), \(j\)\(\leq\)m) in \(V(D)\), denoted as \(a_{ij}=(v_i, v_j)\).

If every arc in directed graph \(D\) is assigned one or more value of real number, the obtained directed graph is called weighted directed graph or directed network, referred to as network, and recorded as \(D=\langle V, A, W \rangle\).

Definition 2: Set \(D=(V, A, W)\) to be a weighted directed graph, \(v_i\) and \(v_j\) to be two vertices in \(D\), then, in all \((v_i, v_j)\) paths of \(D\), path with minimum weight is called the shortest \((v_i, v_j)\) path.

Suppose the network \(D=(V, A, W)\) should not contain negative loop, the vertex set is \(V\)={\(v_1, v_2, \ldots, v_n\)}, firstly, consider the shortest path from the particular vertex \(v_1\) to all other vertices in \(D\), the linear programming model of shortest \((v_1, v_n)\) path is as follows:

\[
\begin{align*}
\text{min} \sum_{(v_i, v_j) \in A} w_{ij}x_{ij} \\
\sum_{(v_i, v_j) \in A} x_{ij} - \sum_{(v_i, v_j) \in A} x_{ji} = \begin{cases} 
1 & i = 1 \\
0 & 2 \leq i \leq n-1 \\
-1 & i = n
\end{cases} \\
x_{ij} = 0 \text{ or } 1 & \forall (v_i, v_j) \in A
\end{align*}
\]

Where, \(w_{ij}\) in condition (1) denotes the weight of arc \((v_i, v_j)\); condition (2) represents that in path \((v_1, v_n)\), in addition to vertex \(v_1\) with only one outgoing -arc and vertex \(v_n\) with only one incoming-arc, all the rest vertices have the same number of outgoing -arc and incoming-arcs; \(x_{ij}\) in condition (3) can take 1 or 0, and they separately represent the times of arc \((v_i, v_j)\) in path \((v_1, v_n)\).

As for a common path \((v_i, v_j)\) (i=1, 2, ..., n), we can make appropriate transformations for conditions (1)~(3) to generate a similar linear programming model. Obviously, if we can solve (1)~(3), the length of shortest \((v_i, v_j)\) path is the value of objective function, and shortest \((v_i, v_j)\) path can be determined according to the value of \(x_{ij}\). Unfortunately, this wasn’t much help. It is usual to translate conditions (1)~(3) into their dual programs, and also translate them to an equation based on the following theorems.

Theorem 1: Set \(D=(V, A, W)\) to be a network without non-plus loop, vertex set is \(V\)={\(v_1, v_2, \ldots, v_n\)}, and there is path \((v_i, v_j)\) (i=1, 2, ..., n) in \(D\), \(\overrightarrow{u_j}\) is the length of shortest \((v_i, v_j)\) (j=1, 2, ..., n) path in \(D\), if, and only if \(\overrightarrow{u_i}=(u_1, u_2, \ldots, u_n)\) meets the following conditions:

\[
\begin{align*}
\overrightarrow{u} = \begin{bmatrix}
0 \\
\min (u_i + w_{ij}) & j = 2, 3, ..., n
\end{bmatrix}
\end{align*}
\]

For convenience, here agreed on \((v_i, v_j) \notin A\), set \(w_{ij} = +\infty\), thus (4) is equivalent to:

\[
\begin{align*}
\overrightarrow{u} = \begin{bmatrix}
0 \\
\min(u_i + w_{ij}) & j = 2, 3, ..., n
\end{bmatrix}
\end{align*}
\]

Equation (5) is called the shortest path equation, which is a non-linear equation group. In order to find \(u_i\), it is necessary to know all the other \(u_i\), so it is very difficult to solve (5) directly. Therefore, it does not seem to do any good when transforming the shortest path problem into (5), but based on it, we can get various algorithms for solving the shortest path from a specific vertex to all other vertices. For special non-loop networks, network with non-negative weight and common non-loop network, the shortest paths from vertex \(v_1\) to all other vertices can be solved respectively by substitution method, Dijkstra algorithm and Floyd algorithm.

B. Optimal path problem

As a classic issue in research on graph theory and network technology, the shortest path problem has been widely used in many fields, including project planning, geographic information systems, communications and military operations research. Therefore, it has important theoretical and applied value to research on the design and improvement of algorithm for solving the shortest path problem. Currently, there are many results of research on the shortest path problem. The traditional algorithms for shortest path problem are mainly Floyd algorithm and Dijkstra algorithm. Among them, Floyd algorithm is a multi-source shortest path algorithm, which is mainly used to calculate the shortest path among all nodes; and Dijkstra algorithm is a single-source shortest path algorithm, which is an efficient algorithm used to calculate the shortest path from a source node to its all places nodes.

The problem of optimal path selection is the core issue of route information function [24-26] in the road-car...
navigation system, which is the problem of seeking the shortest distance between two vertices in a given weighted directed graph based on the shortest path problem in operational research and graph theory. Suppose that \( D = (V, E) \) should be a weighted directed graph, where \( V \) is the set of vertices, and the number of vertices is \( n \), \( E \) is the set of arcs. \( a^w(v_i, v_j) \) represents the arc from \( v_i \) to \( v_j \), \( 1 \leq j \leq n \), \( w(a^w) = w_j \) represents the weight of arc \( a^w(v_i, v_j) \). As for two given vertices \( v_i \) and \( v_j \) in the graph \( D \), suppose \( p \) is a path from \( v_j \) to \( v_i \) of \( D \), the weight value of \( p \) is defined as the sum of the weight of all the arcs of \( p \), denoted by \( w(p) \). The shortest path problem is to find a path which is the minimal weight in all the ways from \( v_j \) to \( v_i \), which is to seek a path \( p_0 \) from \( v_j \) to \( v_i \), and makes \( w(p_0) = \min(w(p)) \) [9].

It should be noted that the shortest path problem described here is broad, that is to say, the variable weight in graph not only refers to the distance between two points. For example, in economic activities, its meaning is very rich; it can be used to express benefits, costs and so on. Therefore, the weights of edge in graph are not all greater than or equal to zero, it may be less than zero. In this study, it is the shortest path routing problem about route choice, so, the weights in graph are all greater than zero. There have been many algorithms for the shortest path problem, but their ideas are all based on Dijkstra algorithm and Floyd algorithm.

Here, we will give an example to specify the above thing. Suppose that \( (v_i, v_j) \) should reach city \( v_k \) from city \( v_i \), as shown in Fig. 1, in the graph, the weight \( w_j \) \((j = 1, 2, \ldots, 12)\) of edge represents the distance between two cities. Because there are a number of transport routes can be chosen from city \( v_i \) to city \( v_k \), then what we need to solve is how to select a right transport route, which makes the distance from \( v_i \) to \( v_k \) be the shortest.

### C. Basic idea of Floyd algorithm

In some problems, it needs to calculate the shortest paths for all the vertex pairs in a given non-negative loop network \( D = (V, A, W) \), where \( V = \{v_1, v_2, \ldots, v_n\} \). And, Floyd algorithm is to solve such problem. Of course, this problem can also be solved by applying substitution method repeatedly, Dijkstra algorithm and Floyd algorithm under certain conditions. For the first time it can solve the shortest path from \( v_1 \) to \( v_2, v_3, \ldots, v_n \). For the second time it can solve the shortest path from \( v_1 \) to \( v_1, v_3, \ldots, v_n \), and for the \( n \)-th time it can solve the shortest path from \( v_n \) to \( v_1, v_2, \ldots, v_{n-1} \). Thus, the shortest paths of all the vertex pairs in \( D \) will be solved. However, if we do like this, the program will be complex and the computation will be large to \( O(n^3) \). And if use Floyd algorithm, the calculation will be simplified.

In the non-negative loop network \( D = (V, A, W) \), for all \((v_i, v_j) \) \( A \), \( w_0 \) is the weight of arc \((v_i, v_j)\), when \((v_i, v_j) \in A \), Floyd algorithm provides:

\[
w_j = \begin{cases} 0 & i = j \\ +\infty & i \neq j \\
\end{cases}
\]

Figure 1. The path map of 5 cities.

Make \( D(i, j, m) \) denotes the sub network of \( D \), which is exported by vertex set \( \{v_i\} \cup \{v_j\} \) \( \{v_1, v_2, \ldots, v_n\} \) \((i, j = 1, 2, \ldots, n)\). In particular, \( D(i, j, 0) \) indicates the sub network of \( D \) exported by vertex set \( \{v_i\} \cup \{v_j\} \). The length of shortest \((v_i, v_j)\) path is denoted as \( u_{ij} \), then obviously there are conclusions as follows:

**Conclusion 1:** \( u_{ij}(0) = w_{ij} \).

**Conclusion 2:** \( D(i, j, n) = D \), so, \( u_{ij}(n) \) is the length of the shortest \((v_i, v_j)\) path in \( D \).

**Conclusion 3:** \( D(i, j, m-1) = D \) is sub-network of \( D(i, j, m) \), therefore \( u_{ij}(m-1) \leq u_{ij}(m) \).

**Conclusions 4:** \( D(i, j, m) = D(i, j, m-1), D(m, m) = D(m, m, m-1), D(m, m-1) = D(m, m, m-1), \) because there are \( u_{im}(m) = u_{im}(m-1) \), \( u_{mj}(m) = u_{mj}(m-1) \), \( u_{mj}(m-1) = u_{mj}(m-1) \).

With the above preparation and several important conclusions, we can draw another very important theorem as follows.

**Theorem 2:** Set network \( D = (V, A, W) \) to do not contain negative loop, then for all \( i, j = 1, 2, \ldots, n \), the length of shortest \((v_i, v_j)\) path in \( D(i, j, m) \) satisfies the following condition:

\[
\begin{align*}
\begin{cases}
\quad u_{ij}^{(0)} = w_{ij} \\
\quad u_{ij}^{(n)} = \min\{u_{ij}^{(n-1)} + u_{mk}^{(n-1)}\} & (1 \leq m \leq n)
\end{cases}
\end{align*}
\]

Theorem 2 is an important basis for Floyd algorithm, making m cycle from 1 to n, then we can solve length \( u_{ij}^{(n)} \) of the shortest \((v_i, v_j)\) path in network \( D \). In order to calculate the shortest \((v_i, v_j)\) path in the network \( D(i, j, m) \) while calculating its length, it needs to introduce a forward tracking method, as follows:

Set \( P_{ij}^{(n)} \) to be the shortest \((v_i, v_j)\) path in network \( D(i, j, m) \), and \( r_{ij}^{(m)} \) denotes the subscript of head of the first arc in \( P_{ij}^{(n)} \), clearly \( r_{ij}^{(0)} = j \). If \( r_{ij}^{(m)} \) has been known, when \( u_{ij}^{(n)} \leq u_{ih}^{(n)} + u_{hj}^{(n)} \), there will be \( u_{ij}^{(n)} = u_{ij}^{(n)} \), hence \( r_{ij}^{(m)} = r_{ij}^{(m)} \), otherwise \( r_{ij}^{(m)} = r_{ij}^{(m)} \). If \( r_{ij}^{(m)} = k \), then the first arc in \( P_{ij}^{(m)} \) is \((v_i, v_k)\), because \( D \) does not contain negative loop, \( P_{ij}^{(n)} \) is the shortest \((v_i, v_j)\) path, therefore, the first arc in \( P_{ij}^{(m)} \) is the second arc in \( P_{ij}^{(m)} \), and so on, all the arcs in \( P_{ij}^{(n)} \) can be determined.

Floyd algorithm also known as plug point method is a kind of dynamic programming [10] used to find the shortest path between vertices of a given weighted graph. The basic idea is:
Suppose, find the shortest path from vertex \( v_i \) to vertex \( v_j \). If there are arcs from the vertex \( v_i \) to vertex \( v_j \), there is a path from vertex \( v_i \) to vertex \( v_j \) whose length is arcs \([i,j]\), the path may not be the shortest path, so \( n \) times tests remain to be done. First, it must be considered whether the path \((v_i, v_0, v_j)\) exists or not. If it exists, then compare the length of path \((v_i, v_j)\) with the length of path \((v_i, v_j, v_k)\), the shorter one is taken as the shortest path from \( v_i \) to \( v_j \), in this path, serial number of the vertexes is not greater than 0. If adding a vertex \( v_i \) in the path, that is, if \((v_i, \ldots, v_j)\) and \((v_1, \ldots, v_j)\) are respectively the shortest paths found currently whose serial numbers of vertexes are not greater than 0, then \((v_i, \ldots, v_j)\) may be the shortest path from \( v_i \) to \( v_j \) and the middle node number is not greater than 1 in the shortest path. It compares with the shortest path from \( v_i \) to \( v_j \) in which the middle node number is not greater than 0, after selecting the shortest path in which the serial number of middle vertex is not greater than 1, \( v_2 \) is added, then continue to test. By analogy with this, through the comparison of \( n \) times, finally, the shortest path from \( v_i \) to \( v_j \) must be obtained.

The Floyd algorithm can be calculated to achieve by weight matrix, that is, beginning with the adjacency matrix \( D^{(0)} \) which is on behalf of the distance of any two nodes from \( v_i \) to \( v_j \), first, calculate \( D^{(1)} \), that is, calculate all of possible paths after a transit from \( v_i \) to \( v_j \), select the shortest path after comparison, replace the corresponding path in \( D^{(0)} \), iteratively list distance matrix \( D^{(j)} \), each element in \( D^{(j)} \) represents the shortest path between any two nodes in the network after an iteration, that is, the shortest path between any two nodes is the shortest one from one node to another node directly or only through an intermediate point in the network. On this basis, in turn calculate \( D^{(2)}, D^{(3)}, \ldots, D^{(k)} \), the corresponding element in \( D^{(k)} \) represents the shortest path between two points without passing through any intermediate point or allowed to pass through the \( 2^k - 1 \) intermediate points. When \( D^{(k+1)}=D^{(k)} \), this shows that the obtained weighted adjacency matrix \( D^{(k)} \) reflects the information of the shortest distance among all the vertices and \( D^{(k)} \) becomes the shortest distance matrix. The algorithm is as follows:

Step1: Make the initial distance matrix \( D^{(0)} = (d^{(0)}_{ij}) \), of which,

\[
d^{(0)}_{ij}=\begin{cases} w_{ij}, & \text{if } v_i, v_j \text{ are adjacent} \\ \infty, & \text{others} \\ \end{cases} (i, j = 1, 2, \ldots, n). (8)
\]

Step2: Construct iterative matrix \( D^{(k)} = (d^{(k)}_{ij}) \), of which,

\[
d^{(k)}_{ij} = \min \{d^{(k-1)}_{ij} + d^{(k-1)}_{ji} \mid i = 1, 2, \ldots, n \}. (9)
\]

Step3: If \( D^{(k+1)}=D^{(k)} \), iteration terminates. Otherwise, go to step2.

Floyd algorithm is a dynamic programming algorithm, which can obtain the shortest path matrix of all the vertex pairs by the weight matrix of graph. The main advantage of this algorithm is easy to be understood; it can obtain the shortest distance between any two nodes; the code is simple; and its core can be completed with only a triple loop. The description by C language is as follows:

for \((k = 0; k < n; k++)\)
{
    for\((i = 0; i < n; i++)\)
    {
        if\((i != k)\)
        {
            for\((j = 0; j < n; j++)\)
            {
                if\((j != i && j != k)\)
                {
                    \&\&(map[i][k] + map[k][j] < map[i][j])
                    \{ 
                        map[i][j] = map[i][k] + map[k][j];
                    \}
                }
            }
        }
    }
}

From the above program segment it can be known that, the time complexity of Floyd algorithm is \(O(n^3)\), it is too high to calculate the shortest path with large amounts. In addition, through analyzing the Floyd algorithm, it is not difficult to find the following problems in the network without negative loop.

(1) When calculating the shortest path between \( v_i \) and \( v_j \), \( n \) times addition is calculated each time, and obviously, intermediate nodes \( v_c \) can not make the path shorter, which reduces the computational efficiency [4].

(2) To find the shortest path between \( v_i \) and \( v_j \), it has to go back to check how to calculate \( d^{(k)}_{ij} \), suppose \( d^{(k)}_{ij} = d^{(k-1)}_{ij} + d_{ij} \), check how to calculate \( d^{(k)}_{ij} \) and \( d^{(k-1)}_{ij} \), like that, ..., find the shortest path until the elements in \( D^{(k)} \) are checked.

Clearly, the method of seeking the shortest path above is more cumbersome, and is not intuitive and not suitable for the calculation of the shortest path in the network with lots of nodes. Based on this, Floyd algorithm has been improved in this paper, in order to reduce the calculation, the method of finding the shortest path becomes more simple and convenient.

III. IMPROVED FLOYD ALGORITHM

A. Improvement Ideas of Floyd Algorithm

As for the two problems of original Floyd algorithm, the following two improvement ideas are proposed.

(1) When constructing the iterative matrix \( D^{(i)} = (d^{(i)}_{ij}) \) to calculate the shortest path between \( v_i \) and \( v_j \), compare the path length of inserted node \( v_c \) first, if \( d^{(i-1)}_{ij} + d_{ic} < d^{(i-1)}_{ij} \), this illustrates that after inserting the node \( v_c \), the length of path from \( v_i \) to \( v_j \) when passing \( v_c \) is not shorter than the original one, so \( d^{(i-1)}_{ij} + d_{ic} \) does not have to be calculated, but search the next node directly.

(2) Construct serial number matrix \( A^{(k)} = (a^{(k)}_{ij}) (k = 0, 1, \ldots) \), which is used to record the circumstance of inserted nodes in the first \( k \)-iteration in the second step of the traditional algorithm.
B Improved Floyd Algorithm

After analyzing for the traditional Floyd algorithm, it is not difficult to find that, when calculating elements \(d_{ij}^{(k)}\) in iterative matrix \(D^{(k)}\), the elements \(d_{pq}^{(k)}\) \((p=1, 2, \ldots, i-1; q=1, 2, \ldots, n)\) of the front \(i-1\) rows in matrix \(D^{(k)}\) and the front \(j-1\) elements \(d_{pq}^{(k)}\) \((q=1, 2, \ldots, j-1)\) of the \(i\)-th row have all been calculated, applying Gauss_Seidel iterative thinking for solving linear equations, these new elements can be used to calculate, and this can speed up the iteration for solving the shortest path.

As \(d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}, d_{ij}^{(k-1)}\}\), when \(r\neq j\), there will be \(d_{ij}^{(k)}\neq d_{ij}^{(k-1)}\); when \(r=i\), there will be \(d_{ij}^{(k)}\neq d_{ij}^{(k-1)}\), therefore \(d_{ij}^{(k)}\) and \(d_{ij}^{(k-1)}\) can be replaced by \(d_{ij}^{(k)}\) and \(d_{ij}^{(k-1)}\) respectively. Thus, the improved Floyd algorithm is gotten, as follows:

Step 1: Make the initial distance matrix \(D^{(0)}=d_{ij}^{(0)}\), where:

\[
d_{ij}^{(0)} = \begin{cases} 
0 & \text{if } j = i = 1, 2, \ldots, n. \\
\infty & \text{otherwise}
\end{cases}
\] (10)

Step 2: Structure the iteration matrix \(D^{(k)}=d_{ij}^{(k)}\), where:

\[
d_{ij}^{(k)} = \min\left\{ d_{ij}^{(k-1)} + d_{ij}^{(k-1)}, d_{ij}^{(k-1)} + d_{ij}^{(k-1)} \right\},
\] (11)

Step 3: If \(D^{(k)}=D^{(k-1)}\), the iteration is end; otherwise, return to step 2.

C Further Optimization of the Improved Floyd Algorithm

For the above improvement ideas, in this study, we further optimized the improved Floyd algorithm.

In fact, if networks are based on non-negative loop, from the analysis of the improved algorithm, it can be known: when calculating the shortest path between two points \(v_i\) and \(v_j\) at a time, it needs to calculate additions \(n\) times, moreover, the inserted intermediate nodes \(v_r\) are many and they can not obviously make the length of path be shorter, which will reduce the computational efficiency, therefore, it can be simplified to lessen operation. To achieve this goal, the improved algorithm is optimized further in this study, the optimization idea is: first, construct iterative matrix \(D^{(k)}=d_{ij}^{(k)}\), when calculating the shortest path between two points \(v_i\) and \(v_j\), compare the length of inserted node \(v_r\) at first, if \(d_{ij}^{(k)}\leq d_{ij}^{(k-1)}\) or \(d_{ij}^{(k-1)}<d_{ij}^{(k)}\), then it denotes that the length of path from \(v_i\) to \(v_j\) through \(v_r\) will not be less than the original length after inserting node \(v_r\), so it does not need to calculate \(d_{ij}^{(k-1)}+d_{ij}^{(k-1)}\), but access to the search of next node directly.

In addition, for the improved algorithm, to find the full path of the shortest path from \(v_i\) to \(v_j\), it usually needs the help of trace-back. That is to say, to find the shortest path between \(v_i\) and \(v_j\), it has to go back to investigate how to calculate \(d_{ij}^{(k)}\). May wish to set \(d_{ij}^{(k)}=d_{ij}^{(k)}+d_{ij}^{(k-1)}\), also check how to calculate \(d_{ij}^{(k)}\), \(d_{ij}^{(k)}\), \ldots, until the elements in \(D^{(0)}\) have been found out, so the shortest path can be obtained. Obviously, the above method to find the shortest path is not intuitive and very complicated. To simplify the calculation, construct a serial number matrix \(A^{(k)}=(a_{ij}^{(k)})\) \((k=0, 1, \ldots)\) to record the case when inserting node in the \(k\)-th iteration in the second step of the traditional algorithm. If set \(a_{ij}^{(k)}=d_{ij}^{(k)}\), then elements in \(a_{ij}^{(k)}\) express the nodes from \(v_i\) to \(v_j\). When the number matrix \(A^{(k)}=a_{ij}^{(k)}\) after the iteration being ended, the full path of the shortest path from \(v_i\) to \(v_j\) is: \(v_i \rightarrow a_{ij}^{(k)} \rightarrow v_j\).

Therefore, the improved Floyd algorithm after being optimized further is as follows:

Step 1: Make the initial distance matrix \(D^{(0)}=d_{ij}^{(0)}\) and serial number matrix \(A^{(0)}=a_{ij}^{(0)}\), of which,

\[
d_{ij}^{(0)} = \begin{cases} 
0 & \text{if } i = j, 1, 2, \ldots, n; \\
\infty & \text{otherwise}
\end{cases}
\] (12)

\[
a_{ij}^{(0)} = \begin{cases} 
0 & \text{if } i = j, 1, 2, \ldots, n; \\
\phi & \text{otherwise}
\end{cases}
\] (13)

At this time, \(d_{ij}^{(0)}\) in \(D^{(0)}\) represents the length of path between \(v_i\) and \(v_j\) without passing other nodes.

Step 2: Construct iterative matrix \(D^{(k)}=d_{ij}^{(k)}\) and serial number matrix \(A^{(k)}=a_{ij}^{(k)}\), of which:

\(D^{(0)}=d_{ij}^{(0)}\) for the element \(d_{ij}^{(k)}\) in \(D^{(0)}\), when \(1 \leq r \leq n\) and \(r \neq i, j\).

If \(d_{ij}^{(k)}=d_{ij}^{(k-1)}\) or \(d_{ij}^{(k)}<d_{ij}^{(k-1)}\), it shows that the length of \(d_{ij}^{(k)}\) of the path will not become shorter after inserting node \(v_r\), \(d_{ij}^{(k-1)}+d_{ij}^{(k-1)}\) does not need to be calculated. Otherwise, let \(d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}+d_{ij}^{(k-1)}, d_{ij}^{(k-1)}+d_{ij}^{(k-1)}\}\), in order to calculate the shortest path between \(v_i\) and \(v_j\) in the \(k\)-th iteration.

Accordingly, all the elements in the serial number matrix \(A^{(k)}\) change as follows:

If \(d_{ij}^{(k)}=d_{ij}^{(k-1)}\) or \(d_{ij}^{(k)}<d_{ij}^{(k-1)}\), points \(v_r\) is recorded, and the value of corresponding element \(a_{ij}^{(k)}\) in \(A^{(k)}\) is changed to \(a_{ij}^{(k)}=\{a_{ij}^{(k-1)}, v_r, a_{ij}^{(k-1)}\}\), this indicates that the length of path from \(v_i\) to \(v_j\) passing \(v_r\) becomes shorter after inserting the node \(v_r\) in this iteration. Otherwise, let \(a_{ij}^{(k)}=d_{ij}^{(k)}\).

Step 3: If \(D^{(k)}=D^{(k-1)}\), the iteration is end; otherwise, return to step 2.

D. Performance Analysis for the algorithm

For the traditional Floyd algorithm, when calculating the shortest path \(d_{ij}^{(k)}\) from \(v_i\) to \(v_j\), it is needed to calculate the sum of elements in \(i\)-th row and the corresponding elements in \(j\)-th column at a time, it is \(n\) times addition totally, and there are \(n^2\) elements in distance matrix. Therefore, the computational complexity of the traditional Floyd algorithm to solve shortest path is \(O(n^2)\).

For the improved Floyd algorithm, firstly, when calculating the shortest path \(d_{ij}^{(k)}\) from \(v_i\) to \(v_j\), it only needs to do \(n^2-2\) times addition at most; Secondly, it has to do comparison each time, that is, if \(d_{ij}^{(k-1)}\leq d_{ij}^{(k-1)}\) or \(d_{ij}^{(k-1)}<d_{ij}^{(k-1)}\), it does not need to calculate \(d_{ij}^{(k-1)}+d_{ij}^{(k-1)}\), in particular, when elements in the \(i\)-th row or \(j\)-th column are all greater than or equal to \(d_{ij}^{(k-1)}\), it can be directly gotten \(d_{ij}^{(k-1)}\), the computation of addition is 0 times. Therefore, in our algorithm, the computation of addition is \(O(n^2)\) times when calculating the shortest
path $d_{ij}^{(k)}$ from $v_i$ to $v_j$, it is a random variable, may wish to be noted for $X$. It is assumed that value of random variable $X$ has equal possibility within the scope of $0 \sim n-2$, and thus the mathematical expectation of random variable $X$ is $E(X) = (n-2)/2$, that is to say, when calculating the shortest path $d_{ij}^{(k)}$ from $v_i$ to $v_j$, the computation of addition is $(n-2)/2$, and there are $n^2$ elements in distance matrix, so, the computational complexity of our algorithm calculating the shortest path is about $O(n^2/2)$. This shows that when calculating shortest path, the computational complexity of our algorithm is half of that of the traditional Floyd algorithm.

However, the analysis of computational complexity for the traditional Floyd algorithm only discussed the computational complexity of the $k$-th iteration, but did not discuss the total required when the algorithm is end. The improved algorithm is an accelerated method for the traditional algorithm, with its advantage reflected by less iteration steps than the traditional algorithm, so iteration steps should be also considered during the algorithm analysis, the initial iteration steps are different because of the difference of the initial distance matrixes, therefore, use computer simulation to illustrate this problem. Use initial distance matrix generated randomly to compare the difference of iteration steps between the traditional algorithm and accelerated algorithm, when seeking the shortest path for any two points. In the test, set the number of vertices to be from 10 to 100, experiment results are as shown in Table 1, which is the comparison of iteration steps of the traditional Floyd algorithm and the improved Floyd algorithm.

From the above experiment, it can be seen that, in the case of less numbers of vertices, compared with the traditional algorithm, the improved algorithm has more advantages, with its total computation being about 27.7% less than that of traditional algorithm.

As for the improved algorithm after being optimized further, before the calculation of $d_{ij}^{(k)}$, it needs comparison each time, that is to say, if $d_{ij}^{(k-1)} \leq d_{k-1}^{(k-1)}$ or $d_{ij}^{(k-1)} < d_{ij}^{(k-1)}$, then it does not need to calculate $d_{ij}^{(k)}$, therefore, the computational complexity of each iteration in the optimized algorithm is less than $O(n^2)$. From the analysis, it is not difficult to find that in the process of calculating $d_{ij}^{(k)}$ in the $k$-th iteration, those points $v_j$ without path to $v_i$, its distance is $d_{ij}^{(k)} = \infty$, which is clearly greater than $d_{ij}^{(k-1)}$, therefore it would not need to calculate $d_{ij}^{(k-1)} + d_{ij}^{(k-1)}$. Thus, when analyzing the computational complexity of the $k$-th iteration, it just needs to consider those points with path to $v_i$, that is edge connected with point $v_i$. May wish to set $m_i$ $(i = 1, 2, \ldots, n)$ to be the number of edges connected with $v_i$, then it needs $m_i$ times operations when calculating $d_{ij}^{(k)}$ (obviously $m_i \leq n$), and there are $n^2$ elements in distance matrix, so the computational complexity of the $k$-th iteration is $\sum_{i=1}^{n} m_i$.

Now, set $m$ to be the average number of edges connected with the nodes in graph, then the computational complexity of the $k$-th iteration is $O(m^2n^2)$. Therefore, for the improved algorithm after being optimized further, the preceding analysis shows that the total computation is even less than that of the algorithm before being optimized.

<table>
<thead>
<tr>
<th>The number of vertices</th>
<th>Traditional Floyd algorithm</th>
<th>Improved Floyd algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>70</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>90</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

IV. THE REALIZATION OF ROUTE SELECTION FUNCTION

A. Algorithm Description

Car navigation system for the driver chooses the optimal path between two locations, corresponding to finding the shortest path between two nodes in a weighted directed graph. Because of the large number of nodes in urban road network, the calculation need to be simplified and efficiency of the algorithm need to be improved in order to select the optimal route for driver quickly. Thus, the improved Floyd algorithm above is used to calculate the shortest path between any two points in urban road network in order to choose the best route for the driver conveniently.

The improved Floyd algorithm was simulated and achieved by using MATLAB, its description of the algorithm is as follows:

Algorithm: ImprovedFloyd.
Input: Urban Road Network $D$.
Output: The length of the shortest path between any two nodes and the shortest paths.

Algorithm Description: The improved Floyd algorithm was used to calculate the shortest path between any two nodes in the urban road network.

1. IF $v_i$ and $v_j$ are adjacent, DO
2. $d_{ij}^{(0)} \leftarrow$ the weight from $v_i$ to $v_j$,
3. $a_{ij}^{(0)} \leftarrow 0$;
4. ELSE $v_i$ and $v_j$ are not adjacent or there is not weight between $v_i$ and $v_j$, DO
5. $d_{ij}^{(0)} \leftarrow \infty$,
6. $a_{ij}^{(0)} \leftarrow \phi$,
7. END
8. $D^{(0)} \leftarrow (d_{ij}^{(0)}), A^{(0)} \leftarrow (a_{ij}^{(0)})$,
9. $k \leftarrow 1$,
10. FOR $i=1 \rightarrow n$ DO
11. FOR $j=1 \rightarrow n$ DO
12. WHILE $r=r-1$; $n$ AND $r \neq i, j$ DO
13. WHILE $D^{(k-1)} \neq D^{(k)}$ DO

© 2010 ACADEMY PUBLISHER
14. IF $d_{ij}^{(k-1)} \leq d_{ij}^{(k-2)}$ OR $d_{ij}^{(k-1)} \leq d_{ij}^{(k-2)}$ DO
15. scan the next node,
16. ELSE DO
17. $d_{ij}^{(k)} = \min \{d_{ij}^{(k-1)}, d_{ij}^{(k-2)} + d_{ij}^{(k-1)}\}$,
18. IF $d_{ij}^{(k)} = d_{ij}^{(k-1)} + d_{ij}^{(k-2)}$ AND $d_{ij}^{(k)} < d_{ij}^{(k-1)}$ DO
19. $d_{ij}^{(k)} = \{d_{ij}^{(k-1)}, v_i, d_{ij}^{(k-1)}\}$,
20. ELSE DO
21. $D^{(k)} = (d_{ij}^{(k)}), A^{(k)} = (a_{ij}^{(k)})$
22. $k \leftarrow k + 1$,

B. Algorithm examples

For example, urban road network is known as shown in Fig. 2, the improved Floyd algorithm is used to calculate the shortest path between any two nodes.

Solution: Obviously, Fig. 2 is non-negative loop network, first of all, according to the step 1 in the improved algorithm, the initial distance matrix $D^{(0)}$ and the serial number matrix $A^{(0)}$ of Figure 1 can be obtained,

$$D^{(0)} = \begin{bmatrix}
0 & 2 & 3 & \infty & \infty \\
1 & 0 & 4 & 3 & \infty \\
\infty & 3 & 0 & 1 & 3 \\
\infty & 1 & \infty & 0 & 1 \\
\infty & \infty & 6 & \infty & 0 \\
\end{bmatrix}$$

$$A^{(0)} = \begin{bmatrix}
0 & 0 & 0 & \phi & \phi \\
0 & 0 & 0 & 0 & \phi \\
\phi & 0 & 0 & 0 & 0 \\
\phi & 0 & \phi & 0 & 0 \\
\phi & \phi & 6 & 0 & \phi \\
\end{bmatrix}$$

Then, $D^{(1)}$ and $A^{(1)}$ are calculated by the initial distance matrix $D^{(0)}$ and the serial number matrix $A^{(0)}$ in accordance with the step 2 in the improved algorithm.

As for $d_{12}^{(0)}$, first conduct a comparison, other elements in first row of $D^{(0)}$ except $d_{12}^{(0)}$ are not less than $d_{12}^{(0)}$, so $d_{12}^{(0)} + d_{23}^{(0)}$ is not calculated no longer, $d_{12}^{(1)} = d_{12}^{(2)} = 2$ can be gotten directly, correspondingly, in the serial number matrix $A^{(0)}$, $d_{12}^{(0)} = d_{12}^{(2)} = 0$ keeps unchanged.

For $d_{13}^{(1)}$, in the first row of $D^{(0)}$, only $d_{12}^{(0)}$ is less than $d_{13}^{(0)}$, so, $d_{13}^{(1)} = \min\{d_{13}^{(0)}, d_{12}^{(0)} + d_{23}^{(0)}\}$, at this time, $d_{13}^{(1)} = d_{13}^{(0)}$, correspondingly, $d_{13}^{(0)} = d_{13}^{(0)} = 0$.

Similarly, other elements of $D^{(1)}$ and $A^{(1)}$ can be calculated.

$$d_{14}^{(1)} = \min\{d_{14}^{(0)}, d_{12}^{(0)} + d_{24}^{(0)}, d_{13}^{(0)} + d_{34}^{(0)}\}$$
$$= \min\{\infty, 5, 4\} = 4$$

$A^{(1)} = v_3$

$D^{(1)} = \begin{bmatrix}
0 & 1 & 2 & 3 & \infty \\
1 & 0 & 4 & 3 & \infty \\
\infty & 3 & 0 & 1 & 3 \\
\infty & 1 & \infty & 0 & 1 \\
\infty & \infty & 6 & \infty & 0 \\
\end{bmatrix}$

$A^{(1)} = \begin{bmatrix}
0 & 0 & 0 & \phi & \phi \\
0 & 0 & 0 & 0 & \phi \\
\phi & 0 & 0 & 0 & 0 \\
\phi & 0 & \phi & 0 & 0 \\
\phi & \phi & 6 & 0 & \phi \\
\end{bmatrix}$

Figure 2. Urban Road Network

© 2010 ACADEMY PUBLISHER
\[ d_{41}^{(1)} = v_2 \]
\[ d_{42}^{(1)} = d_{43}^{(0)} = 1 \]
\[ d_{44}^{(1)} = a_{43}^{(0)} = 0 \]
\[ d_{45}^{(1)} = \min \{ d_{45}^{(0)}, d_{42}^{(0)} + d_{23}^{(0)}, d_{43}^{(0)} \} = \min \{ \infty, 5, 7 \} = 5 \]
\[ a_{41}^{(1)} = v_2 \]
\[ d_{41}^{(0)} = 1 \]
\[ d_{42}^{(0)} = a_{41}^{(0)} = 0 \]
\[ d_{43}^{(0)} = d_{45}^{(0)} = 1 \]
\[ d_{44}^{(0)} = d_{41}^{(0)} = \phi \]
\[ d_{52}^{(0)} = \min \{ d_{52}^{(0)}, d_{53}^{(0)} + d_{32}^{(0)} \} = \min \{ \infty, 9 \} = 9 \]
\[ a_{52}^{(0)} = v_3 \]
\[ d_{53}^{(0)} = d_{55}^{(0)} = 6 \]
\[ a_{53}^{(0)} = 0 \]
\[ d_{54}^{(0)} = \min \{ d_{54}^{(0)}, d_{53}^{(0)} + d_{34}^{(0)} \} = \min \{ \infty, 7 \} = 7 \]
\[ a_{54}^{(0)} = v_3 \]

So, \( D^{(1)} \) and \( A^{(1)} \) can be obtained,

\[
D^{(1)} = \begin{bmatrix}
0 & 2 & 3 & 4 & 6 \\
1 & 0 & 2 & 4 & 4 \\
4 & 2 & 0 & 1 & 2 \\
2 & 1 & 5 & 0 & 1 \\
\infty & 9 & 6 & 7 & 0
\end{bmatrix}
\]

\[
A^{(1)} = \begin{bmatrix}
0 & 0 & 0 & v_5 & v_3 \\
0 & 0 & v_1 & 0 & v_4 \\
v_2 & v_4 & 0 & 0 & v_4 \\
v_2 & 0 & v_2 & 0 & 0 \\
\phi & v_3 & 0 & v_5 & 0
\end{bmatrix}
\]

Obviously, \( D^{(1)} \neq D^{(0)} \), continue iteration, matrix \( D^{(2)} \) and \( A^{(2)} \) can be obtained by using the above method.

\[
D^{(2)} = \begin{bmatrix}
0 & 2 & 3 & 4 & 5 \\
1 & 0 & 2 & 3 & 4 \\
3 & 2 & 0 & 1 & 2 \\
2 & 1 & 3 & 0 & 1 \\
9 & 8 & 6 & 7 & 0
\end{bmatrix}
\]

\[
A^{(2)} = \begin{bmatrix}
0 & 0 & 0 & v_5 & v_3 \\
0 & 0 & v_1 & v_3 & v_4 \\
v_2 & v_4 & 0 & 0 & v_4 \\
v_2 & 0 & v_2 & 0 & 0 \\
v_4 & v_5 & 0 & v_5 & 0
\end{bmatrix}
\]

After comparison, it is not difficult to find that \( D^{(2)} \neq D^{(1)} \), then calculate \( D^{(3)} \) according to \( D^{(2)} \).

\[
D^{(3)} = \begin{bmatrix}
0 & 2 & 3 & 4 & 5 \\
1 & 0 & 2 & 3 & 4 \\
3 & 2 & 0 & 1 & 2 \\
2 & 1 & 3 & 0 & 1 \\
9 & 8 & 6 & 7 & 0
\end{bmatrix}
\]

Through comparison, we can see \( D^{(3)} = D^{(2)} \), algorithm terminated. So, \( d_{ij}^{(2)} \) of \( D^{(2)} \) is the value of length of the shortest path between \( v_i \) and \( v_j \), \( a_{ij}^{(2)} \) of \( A^{(2)} \) is the shortest path between \( v_i \) and \( v_j \). For example, if the length of the shortest path and the shortest path are demanded to calculate between node \( v_2 \) and node \( v_4 \), search corresponding \( d_{24}^{(2)} \) and \( a_{24}^{(2)} \) of matrix \( D^{(2)} \) and matrix \( A^{(2)} \), it can be obtained that the length of the shortest path is 3, the shortest path is \( v_2 \rightarrow v_3 \rightarrow v_4 \).

V. CONCLUSION

Route selection plays an important role in car navigation system, and there are a large number of nodes in urban road network, if use the traditional Floyd algorithm to calculate the shortest path, each node in the network must be involved, so the computation will be very large.

For the above problem, the traditional Floyd algorithm was improved and optimized further by constructing iterative matrix and serial number matrix, and the improved algorithm was successfully used to solve route selection problem in car navigation system. Those inserted nodes not directly connected to the node pair can not make the length of path shorter. There are also some nodes directly connected to the node pair, but the length of path after inserting is longer than before. In the improved algorithm, make comparison before calculating, and the above two kinds of nodes would not be involved in calculation. Therefore, the computation will be reduced largely. From the comparison with the traditional Floyd algorithm, the computational complexity of the improved Floyd algorithm has been reduced to half of that of the traditional algorithm. In addition, while calculating the
shortest path, use serial number matrix to record the nodes making the path shorter, this will make the solving for the shortest path more simple, convenient and direct.

Overall, the improved Floyd algorithm not only has less complexity but also has more effectiveness in the process of searching the shortest path.

REFERENCES

[10] W. Wei. “Research and implementation of the shortest path algorithm based on real time traffic information”, Chang’an University, Xi’an, China, 2009.