International Journal of Physics and Mathematical Sciences ISSN: 2277-2111 (Online)
An Open Access, Online International Journal Available at http://www.cibtech.org/jpms.htm
2016 Vol. 6 (1) January-March, pp. 24-36/Mao and Shi.
Research Article

THE APPLICATION OF ARTICLE *k-TH* SHORTEST TIME PATH ALGORITHM

Hua Mao and *Ming Shi
Department of Mathematics and Information Science, Hebei University, Baoding 071002, China
**Author for Correspondence*

ABSTRACT
If a person wants to travel to a place by some kinds of transportation, he (or she) will consider an optimal path, which costs a shortest time among all paths. However, when a situation is not expected to happen, such as a heavy rain, the person will have to select the second, the third… the *k-th* shortest path to continue his (or her) travel. In this paper, for searching the *k-th* path, we give an improved Floyd algorithm by constructing the iterative matrix and ordinal matrix for solving the shortest path in the directed graph and undirected graph. The *k-th* shortest path algorithm can not only calculate the shortest path weights more quickly, but also find a shortest path more directly. Our algorithm gives a more accurate time to judge a travel plan. We use our algorithm, the iteration speed, the amount of computation is reduced to a certain degree. Two examples are given to illustrate the superiority of our algorithm.

Keywords: *Directed Graph and Undirected Graph, the Shortest Time Path, k-th Shortest Time Path Algorithm*

INTRODUCTION
With the development of society, people pay much more attention to the effective use of time. In order to save the time on traveling by vehicles, it is necessary for a person to estimate the time spent on the road and to choose the route that costs the least time (Yan and Liu, 2000).

Generally, a plan has been given in advance, but weather disasters or other no expected events always make the plan not run properly. Under this case, how to choose the second path, or the *k-th* path ($k \geq 2$) that is a problem. Up to now, many algorithms have been provided for solving the shortest path, such as Dijksta algorithm which proposed by Dick Stella, a computer scientist from the Netherlands, in 1959. Classical Floyd algorithm was proposed by Floyd in 1962. The readers can refer to (Wang, 2009). In addition, many people have studied *k-th* shortest path algorithm from different angles. Eppstein (1999) gave an algorithm for choosing a path on the directed graph which allowed the existence of rings. Li (2006) gave a new *k-th* shortest path algorithm in an undirected graph. Classical Floyd algorithm can easily find the shortest path between any two vertexes on a graph, but not directly reflect the sequence of the shortest path. Dai and Cheng (2013) introduced an improved Floyd algorithm, using C++ language programming technology, which reflects the relationship between any two vertices of the shortest path sequence. Even though, the classical Floyd algorithm still needs to give new improvement.

In order to give a new improvement for the classical Floyd algorithm, we need to find and overcome some defects in classical Floyd algorithm. This paper will do these works. More details are as follows:

(1) We will give a weight for every vertex. This weight stands for residence time on this vertex. Travel staff knows their travel plan and the residence time needed by the site. It is very important to improve the effective use of time.

(2) When we have obtained a vertex, we need to search the next vertices by classical Floyd algorithm. During this time, we just provide an idea that should compare the values of the distance between the obtained vertices and any of next vertices. The purpose of this improvement is to delete any of unnecessary vertices in order to save the searching time. Repeating the process described in the above (2) with deleting some arcs which have not new vertices connected with, we can find the *k-th* optimal route which we hope to obtain.

Preliminaries

According to the symbol given by Wang *et al.*, (2010), some symbols used in the paper are given as follows:

W represents directed graph and undirected graph weighted matrix;

$v0$ and vt represents an arbitrary source vertex and target vertex respectively;

p is the shortest path to the source point v_i to the target point v_j $(1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n)$;

d is the weight matrix of the shortest path;

w_{kk} represent the residence time of each vertex and $w_{kk} \neq 0$ $(1 \leqslant k \leqslant n)$;

p1 is the shortest time path of the reverse p;

tt is changing of vertex on the path;

Inf expresses ∞.

Some of the definitions used in this paper are derived from (Bondy and Murty, 2008).

Definition 1 A graph *G* is an ordered pair *(V(G),E(G))* consisting of a set *V(G)* of vertices and a set *E(G)*, disjoint from *V(G)*, of edges, together with an incidence functions that associates with each edge of *G* an unordered pair of (not necessarily distinct) vertices of *G*. If *e* is an edge and *u* and *v* are vertices such that, Then *e* is said to join *u* and *v*, and the vertices *u* and *v* are called the ends of *e*. We denote the numbers of vertices and edges in *G* by *v(G)* and *e(G)*, these two basic parameters are called the order and size of *G*, respectively.

Definition 2 Let *G* be a graph, with vertex set *V* and edge set *E*. The incidence matrix of *G* is the $n \times m$ matrix $M_G := (m_{ve})$, where m_{ve} is the number of times (0, 1, or 2) that vertex *v* and edge *e* are incident.

Clearly, the incidence matrix is just another way of specifying a graph.

Algorithm Idea

This section will first point the flaws in classical Floyd algorithm, and second give our improvement idea for classical Floyd algorithm. For the classical Floyd algorithm, the reader can refer to Bondy and Murty, (2008).

Floyd Algorithm Defects

We can find the defects existed in the classical Floyd algorithm.

Defect 1

For travelers, it is necessary to calculate the accurate time required for their travels. But in the classical Floyd algorithm, a traveler cannot find the residence time at every vertex. Hence, classical Floyd algorithm cannot reflect the shortest time path directly.

Defect 2

Using classical Floyd algorithm to find the *k-th* shortest path, a traveler needs to calculate the shortest path between two vertexes v_i and v_j each time to calculate the *n* (here *n* is the number of vertices in the directed graph and undirected graph.) addition, and the insertion of the middle vertex v_k is obviously the length of the path is not be shortened, reducing the computational efficiency. Classical Floyd algorithm is described in the reference (Xie and Li, 1995). When we calculate the large-scale path, the times for calculating becomes bigger.

Improvement Idea

In order to overcome Defects 1 and 2, we will give the idea of our algorithm as follows. According to the designing a plan of a travel, the directed graph and undirected graph is set up. Based on this graph, the classical Floyd algorithm can run. Hence, we will improve classical Floyd algorithm with the directed graph and undirected graph.

To improved Defect 1.

Based on the directed graph and undirected graph, for each vertex, we give a weight w_{kk} satisfying $w_{kk} \neq 0$, where w_{kk} is the residence time on vertex *k*. It can be achieved in the judgment time accuracy.

To improved Defect 2.

When calculating the short path between two vertices v_i and v_j, we should treat the insertion of the vertex v_k comparison the length of the path first. If $d_{ik}^{(t)} \geqslant d_{ij}^{(t)}$ or $d_{kj}^{(t)} \geqslant d_{ij}^{(t)}$ or $w_{kk} \geqslant d_{ij}^{(t)}$, $(t = 1,2,...n)$, then the length of vertex v_i passes through the vertex v_k to reach the vertex v_j not shorter than the original. So, we no longer need to calculate $d_{ik}^{(t)} + d_{kj}^{(t)}$, and will find the next vertex during the search. This determination will delete many unnecessary calculate vertices. All of these vertices are not involved in calculation. Thus, this improvement will greatly reduce the amount of calculation.

Algorithm Process

In the above section, we have introduced the idea of our algorithm.

Firstly, this section will provide the process of our algorithm using Matlab language. Secondly, we will analyze the complexity of our algorithm. Thirdly, we will compare the properties between our algorithm and classical Floyd algorithm.

First, we provide some notations used in Matlab.

p represents the path matrix

d represents the shortest weight path matrix

Second, because our algorithm needs to call improved Floyd algorithm (or say, prepared algorithm), so we present an algorithm as a prepared process for our algorithm.

Prepared Algorithm

Input: The directed graph and undirected graph matrix W; the source point $v0$; the target point vt.

Output: The source point $v0$ to the target point vt and the shortest path $d(v0, vt)$.

With Matlab language, the concrete process is as follows:

Matlab procedures are described as follows

```
function floyd(w,v0,vt)
n=length(w);
d=w;
k=1;
for i=1:n
   d(i,i)=0;
end
while k<=n
   for i=1:n
      for j=1:n
         if d(i,j)>d(i,k)&&d(i,j)>d(k,j)&&d(i,j)>w(k,k)&&i~=k&&k~=j
            %iterative shortest path d(i,j)
            d(i,j)=min(d(i,j),d(i,k)+d(k,j)+w(k,k));
         end
      end
   end
   k=k+1;
end
k=1;
p1(k)=vt;
%shortest path target vertex vt
tt=vt;
%tt=v0the shortest path to the end of the search
while tt~=v0
    for j=1:n
       if d(v0,tt)==w(v0,j)
          k=k+1;
          p1(k)=v0;
          tt=v0;
          break;
       else if v0~=j&&d(v0,j)==d(v0,tt)-w(j,tt)-w(j,j)
          k=k+1;
%the record of the shortest path number (reverse)
          p1(k)=j;
          tt=j;
 break;
       end
    end
```

```
end
k=1;
r=find(p1~=0);
t=length(r);
for j=t:(-1):1
%the record of the shortest path number (positive)
p(k)=p1(r(j));
   k=k+1;
end
disp(d)
disp(d(v0,vt))
disp(p)
```

The k-th Shortest Path Algorithm

Our algorithm is to delete some arcs and call the prepared algorithm to calculate the *k-th* shortest path.

Input: the directed graph and undirected graph weighted matrix W, the source point $v0$, the target point vt.

Output: Source point $v0$ to target point vt, shortest path $dk(v0,vt)$ and the shortest path pk.

Matlab procedures are described as follows

p1, p2, p3, p4, p5, pk represent the first, second, third, fourth, fifth, the *k-th* shortest path respectively.

d1, d2, d3, d4, d5, dk represent the length of the p1, p2, p3, p4, p5, pk respectively.

a represents weighted graph matrix.

Step 1: Calculation of the Second Shortest Path Algorithm

Second path algorithm

```
function [d2,p2]=k2th(w,v0,vt)
n=length(w);
[d1,p1]=floyd(w,v0,vt);% for the most short circuit
num1=length(p1);
d2=inf;
for i=1:(num1-1)
   a=w;
   a(p1(i),p1(i+1))=inf; % delete an edge on the shortest way
   a(p1(i+1),p1(i))=inf;
   [d,path]=floyd(a,v0,vt); % recalculate the short circuit
   if d<d2
      d2=d;
      p2=path;
   end
end
```

Step 2: Calculation of the Third Shortest Path Algorithm

Matlab procedures are described as follows

```
function [d3,p3]=k3th(w,v0,vt)
[d1,p1]=floyd(w,v0,vt);
[d2,p2]=k2th(w,v0,vt);
num1=length(p1);
num2=length(p2);
d3=inf;
for i=1:(num1-1)
   for j=1:(num2-1)
```

```
        a=w;
        a(p1(i),p1(i+1))=inf;
        a(p1(i+1),p1(i))=inf;
        a(p2(j),p2(j+1))=inf;
        a(p2(j+1),p2(j))=inf;
[d,path]=floyd(a,v0,vt);
        if d<d3
           d3=d;
           p3=path;
        end
    end
end
```

Step 3: Calculation of the Fourth Shortest Path Algorithm

```
function [d4,p4]=k4th(w,v0,vt)
[d1,p1]=floyd(w,v0,vt);
[d2,p2]=k2th(w,v0,vt);
[d3,p3]=k3th(w,v0,vt);
num1=length(p1);
num2=length(p2);
num3=length(p3);
d4=inf;
for i=1:(num1-1)
    for j=1:(num2-1)
        for k=1:(num3-1)
            a=w;
            a(p1(i),p1(i+1))=inf;
            a(p1(i+1),p1(i))=inf;
            a(p2(j),p2(j+1))=inf;
            a(p2(j+1),p2(j))=inf;
            a(p3(k),p3(k+1))=inf;
            a(p3(k+1),p3(k))=inf;
[d,path]=floyd(a,v0,vt);
            if d<d4
               d4=d;
               p4=path;
            end
        end
    end
end
```

Step 4: Calculation of the Fifth Shortest Path Algorithm

```
function [d5,p5]=k5th(w,v0,vt)
[d1,p1]=floyd(w,v0,vt);
[d2,p2]=k2th(w,v0,vt);
[d3,p3]=k3th(w,v0,vt);
[d4,p4]=k4th(w,v0,vt);
num1=length(p1);
num2=length(p2);
```

```
num3=length(p3);
num4=length(p4);
d5=inf;
for i=1:(num1-1)
   for j=1:(num2-1)
      for k=1:(num3-1)
         for l=1:(num4-1)
            a=w;
            a(p1(i),p1(i+1))=inf;
            a(p1(i+1),p1(i))=inf;
            a(p2(j),p2(j+1))=inf;
            a(p2(j+1),p2(j))=inf;
            a(p3(k),p3(k+1))=inf;
            a(p3(k+1),p3(k))=inf;
            a(p4(l),p4(l+1))=inf;
            a(p4(l+1),p4(l))=inf;
[d,path]=floyd(a,v0,vt);
            if d<d5
               d5=d;
               p5=path;
            end
         end
      end
   end
end
…
```

Step k-1. We will obtain the *k*-1-*th* shortest path as the similar process to the *k-th* shortest path algorithm in Step 1.

Step k. ***Calculation of the k-th Shortest Path***

1. Delete the *k*-1path, and get the weight graph $w_k^{(i,jL)}$;

2. Calculate the shortest path weight $d^{(i,jL)}$ and the shortest $path^{(i,jL)}$ in the weight graph $w_k^{(i,jL)}$;

3. Take $dk = \min\{d^{(i,jL)}\}$, and get the correspondent path *pk*.

Complexity Analysis

This subsection will analyze the complexity for the above algorithms.

For the prepared algorithm, we only need to calculate the *n*-2 times in the calculation of the vertex v_i to the vertex v_j of the shortest path $d_{ij}^{(t)}$ because $k \neq i, j$. Before the calculation of $d_{ij}^{(t)}$, every time you want to compare, that if $d_{ik}^{(t-1)} \geqslant d_{ij}^{(t-1)}$, $d_{kj}^{(t-1)} \geqslant d_{ij}^{(t-1)}$, $d_{ij}^{(t-1)} \geqslant w_{kk}^{(t-1)}$ is without calculation $d_{ik}^{(t-1)} + d_{kj}^{(t-1)}$. Especially, when the *i-th* row or the *j-th* column element is greater than or equal to $d_{ij}^{(t-1)}$, at this time, we obtain $d_{ij}^{(t)} = d_{ij}^{(t-1)}$, addition calculation is 0. In the prepared algorithm, the computation of the shortest path $d_{ij}^{(t)}$ the vertex is a random variable, recorded as X. Now, suppose that a random variable X value in 0: *n*-2 is possible, resulting in random variable mathematical expectation is $\frac{n-2}{2}$, namely in the calculation of the

shortest $d_{ij}^{(t)}$ in the vertex v_i to a vertex v_j, addition amount of calculation for $\frac{n-2}{2}$ and distance matrix in n^2 elements, so the complexity of the algorithm is about $O\left(\frac{1}{2}n^3\right)$. Because the weight of each matrix is not the same, so add three judgments, the possible existence of two cases: when the three judge conditions cannot delete search vertices, as was the complexity of the algorithm and the classical Floyd algorithm; when the three judge conditions can be deleted to find the vertex, the complexity of the algorithm will be far below the classical Floyd algorithm.

As a result of the three judgment conditions, we delete an unnecessary calculation vertex, resulting in a decrease in the number of cycles.

Using prepared algorithm, we can obtain the1st shortest path, the time complexity is about $O\left(n^3\right)$.

Using the *k-th* shortest path algorithm of Step 1, we can obtain the 2nd shortest path, the time complexity is about $O\left(n-1\right)^3$.

Using the *k-th* shortest path algorithm of Step 2, we can obtain the 3rd shortest path, the time complexity is about $O\left(n-2\right)^3$.

…

Using the *k-th* shortest path algorithm of Step *k*, we can obtain the *k-th* shortest path, the time complexity is about $O\left(n-3\right)^3$.

Thus, the complexity of obtaining the *k-th* shortest path is about $O\left(n-k\right)^3$.

Comparision

This subsection will compare some properties between our algorithm provided above and classical Floyd algorithm from the three facts: storage space, judging travel time in accuracy and the complexity of time algorithm. After that, we will use a table to sum up the two algorithms.

1) Storage Space

In our algorithm, we give three conditions $d_{ij}^{(t)} > d_{ik}^{(t)}$, $d_{ij}^{(t)} > w_{kk}^{(t)}$, $d_{ij}^{(t)} > d_{kj}^{(t)}$ for judgment. So, when looking for the weight time matrix, the computation time of our algorithm will be smaller. That is, *m* (*m* represents the number of vertices of the matrix after the deletion of the vertex, and *n* is the number of the matrix vertices) When three conditions are added to delete the vertices, the number of cycles is obviously reduced.

When the three conditions cannot be satisfied, namely $m=n$, the complexity of our algorithm and the classical Floyd algorithm are the same; When three conditions can delete some vertexes, namely $m<n$. The time complexity of the algorithm is less than that the classical Floyd algorithm.

2*) Judging Travel Time in Accuracy*

Because people′s work schedule is very tight, the time for the expected travel arrangements also require more accurate. It requires a precise time to judge a travel plan.

In our algorithm, for each vertex *k*, we give a weight. This follows that our result is the closer to the people′s hope.

Using classical Floyd algorithm, travelers cannot achieve the accurate judgment. Classical Floyd algorithm will not be more accurate to close to people′s travel time.

3) Complexity of Time Algorithm

According to the subsection complexity analysis, we find that the complexity of our algorithm is the same as that in classical Floyd algorithm under $m=n$, and is not the same under $m \neq n$.

From the above three points 1), 2) and 3), we may obtain the following Table 1.

Table 1: Comparison of the Algorithm in this Paper and Classical Floyd Algorithm

Property	Algorithm in this Paper		Classical Floyd Algorithm
Whether Judge Travel Time in Accuracy	Yes		No
Storage Space	$m=n$	$m<n$	$m=n$
	the Same	Reduce	the Same
Complexity of Time Algorithm	$O(m^3)=O(n^3)$	$O(m^3)<O(n^3)$	$O(m^3)=O(n^3)$

Remark: In our algorithm, there are three conditions for deleting vertices. The first vertex cannot be connected directly to the vertex. The second insert vertex cannot make the length of the path shorter. The third vertex weight is longer than the distance of the two vertices. The vertices are not involved in calculation, thus reducing the amount of calculation greatly.

Example

We provide two examples to verify the feasibility of our algorithm in directed graph and undirected graph, respectively.

Example 1

In this paper, we use the example as that in Zhang and Wu (2009) to calculate. Our algorithm is joined the weights of the vertex. Hence, for each vertex in the original graph, we give a weight as v1=3, v2=4, v3=3, v4=4, v5=3.

A directed graph is shown in Figure 1. We hope to calculate the shortest path among all vertexes.

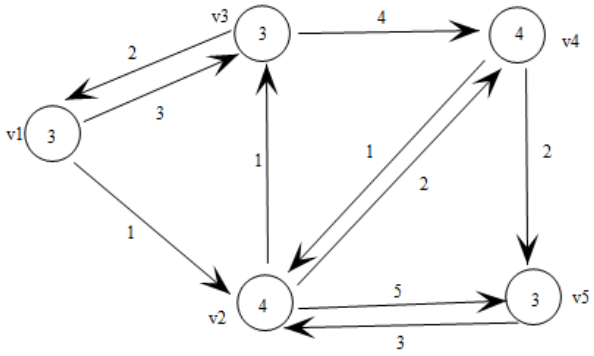

Figure 1 A Directed Graph

Solution:

From Figure 1, we initial weighted distance matrix $D^{(0)}$ and serial number matrix $A^{(0)}$ as the following.

$$D^{(0)}=\begin{bmatrix} 3 & 1 & 3 & \infty & \infty \\ \infty & 4 & 1 & 2 & 5 \\ 2 & \infty & 3 & 4 & \infty \\ \infty & 1 & \infty & 4 & 2 \\ \infty & 3 & \infty & \infty & 3 \end{bmatrix}, A^{(0)}=\begin{bmatrix} 0 & 0 & 0 & \phi & \phi \\ \phi & 0 & 0 & 0 & 0 \\ 0 & \phi & 0 & 0 & \phi \\ \phi & 0 & \phi & 0 & 0 \\ \phi & 0 & \phi & \phi & 0 \end{bmatrix};$$

Matrix $D^{(1)}$ and serial number matrix $A^{(1)}$ are calculated by using initial weighted distance matrix $D^{(0)}$ and serial number matrix $A^{(0)}$ according to the Step 2 in prepared algorithm.

For iterative matrix $d_{12}^{(1)}$, since the first row in the iterative matrix $D^{(0)}$ elements are not less than iterative matrix $d_{12}^{(0)}$ element, so do not calculate $d_{1k}^{(0)}+d_{k2}^{(0)}$, directly get $d_{12}^{(1)}=d_{12}^{(0)}=1$. The corresponding serial number matrix is $A^{(1)}$, and then $a_{12}^{(1)}=a_{12}^{(0)}$ remains unchanged.

For iterative matrix $d_{13}^{(1)}$, because the value of the first element in the first row of the iterative matrix $D^{(0)}$ is smaller than the $d_{13}^{(0)}$ element, it is only $d_{12}^{(0)}$.

Therefore, there is $d_{13}^{(1)}=\min\left\{d_{13}^{(0)},d_{12}^{(0)}+d_{23}^{(0)}+w_{22}\right\}=\min\left\{3,1+1+1\right\}=3$. At this time, we decide $v_1=v_2$.

The corresponding serial number matrix is $A^{(1)}$, and there is $a_{13}^{(1)}=v_2$.

Similarly, the other elements in $D^{(1)}$ and $A^{(1)}$ are calculated as follows:

$d_{14}^{(1)}=\min\left\{d_{14}^{(0)},d_{12}^{(0)}+d_{24}^{(0)}+w_{22},d_{13}^{(0)}+d_{34}^{(0)}+w_{33}\right\}=\min\left\{\infty,1+2+4,3+4+3\right\}=7$, $a_{14}^{(1)}=v_2$;

$d_{15}^{(1)}=\min\left\{d_{15}^{(0)},d_{12}^{(0)}+d_{25}^{(0)}+w_2\right\}=\min\left\{\infty,1+5+4\right\}=10$, $a_{15}^{(1)}=v_2$;

$d_{21}^{(1)}=\min\left\{d_{21}^{(0)},d_{23}^{(0)}+d_{31}^{(0)}+w_{33}\right\}=\min\left\{\infty,1+2+3\right\}=6$, $a_{21}^{(1)}=v_3$;

$d_{23}^{(1)}=d_{23}^{(0)}=1, a_{23}^{(0)}=0$;

$d_{24}^{(1)}=d_{24}^{(0)}=2, a_{24}^{(1)}=0$;

$d_{32}^{(1)}=\min\left\{d_{32}^{(0)},d_{31}^{(0)}+d_{12}^{(0)}+w_{11},d_{34}^{(0)}+d_{42}^{(0)}+w_{44}\right\}=\min\left\{\infty,2+1+3,4+1+4\right\}=6$, $a_{32}^{(1)}=v_1$;

$d_{34}^{(1)}=d_{34}^{(0)}=4, a_{34}^{(1)}=0$;

$d_{35}^{(1)}=\min\left\{d_{35}^{(0)},d_{34}^{(0)}+d_{45}^{(0)}+w_{44}\right\}=\min\left\{\infty,4+2+4\right\}=10$, $a_{35}^{(1)}=v_4$;

$d_{41}^{(1)}=d_{41}^{(0)}=\infty, a_{41}^{(1)}=\phi$;

$d_{42}^{(1)}=d_{42}^{(0)}=1, a_{42}^{(1)}=0$;

$d_{43}^{(1)}=\min\left\{d_{43}^{(0)},d_{42}^{(0)}+d_{23}^{(0)}+w_{22}\right\}=\min\left\{\infty,1+1+4\right\}=6$, $a_{43}^{(1)}=v_2$;

$d_{45}^{(1)}=d_{45}^{(0)}=2, a_{45}^{(1)}=\phi$;

$d_{51}^{(1)}=d_{51}^{(0)}=\infty, a_{51}^{(1)}=\phi$;

$d_{52}^{(1)}=d_{52}^{(0)}=3, a_{52}^{(1)}=0$;

$d_{53}^{(1)}=\min\left\{d_{53}^{(0)},d_{52}^{(0)}+d_{23}^{(0)}+w_{22}\right\}=\min\left\{\infty,3+1+4\right\}=8$, $a_{53}^{(1)}=v_2$;

$d_{54}^{(1)}=\min\left\{d_{54}^{(0)},d_{52}^{(0)}+d_{23}^{(0)}+w_{22}\right\}=\min\left\{\infty,3+1+4\right\}=8$, $a_{54}^{(1)}=v_2$;

So, we get the matrix $D^{(1)}$ and $A^{(1)}$ are as follows.

$$D^{(1)}=\begin{bmatrix}0&1&3&7&10\\6&0&1&2&5\\2&6&0&4&10\\\infty&1&6&0&2\\\infty&3&8&8&0\end{bmatrix},\ A^{(1)}=\begin{bmatrix}0&0&v_2&v_2&v_2\\v_3&0&0&0&v_4\\0&v_1&0&0&v_4\\\phi&0&v_2&0&0\\\phi&0&v_2&v_2&0\end{bmatrix}.$$

Clearly, we have $D^{(1)} \neq D^{(0)}$. This needs to continue iteration. Using the above method, we get the matrix $D^{(2)}$ and $A^{(2)}$ are the following:

$$D^{(2)} = \begin{bmatrix} 0 & 1 & 3 & 7 & 10 \\ 6 & 0 & 1 & 2 & 5 \\ 2 & 6 & 0 & 4 & 10 \\ 11 & 1 & 6 & 0 & 2 \\ 13 & 3 & 8 & 9 & 0 \end{bmatrix}, A^{(2)} = \begin{bmatrix} 0 & 0 & v_2 & v_2 & v_2 \\ v_3 & 0 & 0 & 0 & v_4 \\ 0 & v_1 & 0 & 0 & v_4 \\ v_2, v_3 & 0 & v_2 & 0 & 0 \\ v_2, v_3 & 0 & v_2 & v_2 & 0 \end{bmatrix}.$$

The comparison is not difficult to find $D^{(2)} \neq D^{(1)}$, and then it calculates $D^{(3)}$ according to the $D^{(2)}$.

$$D^{(3)} = \begin{bmatrix} 0 & 1 & 3 & 7 & 10 \\ 6 & 0 & 1 & 2 & 5 \\ 2 & 6 & 0 & 4 & 10 \\ 11 & 1 & 6 & 0 & 2 \\ 13 & 3 & 8 & 9 & 0 \end{bmatrix}.$$

By comparison, we can know $D^{(3)} = D^{(2)}$ by algorithm termination. The corresponding element $a_{ij}^{(2)}$ in $A^{(2)}$ is the shortest path between vertex v_i and vertex v_j. So the value of the corresponding element $d_{15}^{(2)}$ in $D^{(2)}$ is the shortest path length between vertex v1 and vertex v5.
For example, we seek the shortest path time and shortest path between vertex v1 and vertex v5.

Look for the elements $d_{15}^{(2)}$ and $a_{15}^{(2)}$ corresponding to the matrix $D^{(2)}$ and $A^{(2)}$.

When we encounter a path from vertex v1 to vertex v5, we cannot go, then we take this path to delete. We make some of the elements in the matrix $D^{(0)}$ into ∞. Here we give the second shortest time path.
We can use the improved Floyd algorithm to work out the first path. The first shortest time path is v1-v2-v5, the required time is 5. When the first path v2 to v5 in this road cannot go, we make elements in the matrix $d_{25}^{(0)} = \infty$.

We initial weighted distance matrix $D^{(0)}$ and serial number matrix $A^{(0)}$ as follows.

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

Then use the improved Floyd algorithm to calculate, you can get the second shortest time path is v1-v2-v4-v5, the required time is 13;
Then we delete the shortest time path is v1-v2-v4-v5. We make elements in the matrix $d_{24}^{(0)} = \infty$, $d_{25}^{(0)} = \infty$.
Then use the improved Floyd algorithm to calculate, you can get the third shortest time path is v1-v3-v4-v5, the required time is 16;
Then we delete the shortest time path is v1-v3-v4-v5. We make elements in the matrix $d_{24}^{(0)} = \infty$, $d_{25}^{(0)} = \infty$, $d_{13}^{(0)} = \infty$.

Then use the improved Floyd algorithm to calculate, you can get the fourth shortest time path is v1-v2-v3-v4-v5, the required time is 19;

Then we delete the shortest time path is v1-v2-v3-v4-v5. We make elements in the matrix $d_{12}^{(0)} = \infty$, $d_{45}^{(0)} = \infty$,,

Then use the improved Floyd algorithm to calculate, you can get the fifth shortest time path is v1-v3-v4-v2-v5, the required time is 24.

Example 2

We according to the reference (Wang *et al*., 2010, pp.32) case in 2.5, to the editor gives a figure close to the actual circumstance of undirected weighted time graph. Figure 2 shows the graph of a city's subway line.

A traveler wants to travel by subway. Here we design a travel plan, He (She) wants from the vertex v2 to vertex v11. However, when a path cannot travel, the next second, third, fourth, fifth of the shortest path how to go? What is the shortest time to the shortest path?

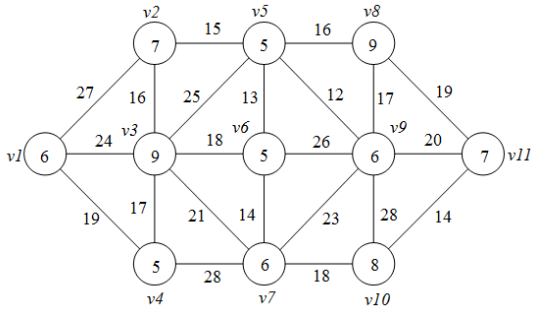

Figure 2: Undirected Weighted Time Graph

Using our algorithm, we calculated the accuracy of shortest time and the corresponding shortest path. Five kind of shortest time path are shown in the following:

vertex v2 to vertex v11

The first shortest time path is v2-v5-v9-v11, the required time is 62;

The second shortest time path is v2-v5-v9-v8-v11, the required time is 87;

The third shortest time path is v2-v3-v7-v10-v11, the required time is 92;

The fourth shortest time path is v2-v5-v9-v10-v11, the required time is 92;

The fifth shortest time path is v2-v3-v5-v9-v11, the required time is 97.

As can be seen from the above results, we can use our algorithm to find the shortest path of any two vertices in the graph and the shortest time to spend. In addition, by using our algorithm, we can estimate more accurate travel time for travelers than using the classical Floyd algorithm. So, our algorithm is more effective for travelers than the classical Floyd algorithm.

Based on the two examples above, we obtain the comparison of the cycle times of our algorithm and the classical Floyd algorithm.

Table 2: Cycle Times

Cycle Times / Vertices	Our Algorithm	Classic Floyd Algorithm
5	17	125
11	186	1331

Table 2 shows the number of cycle times of our algorithm and the classical Floyd algorithm for searching the *k-th* shortest path. In this experiment, two examples of the cycle times are much lower than the classic

Floyd algorithm, and it means that the time efficiency of our algorithm is higher than that of the classical Floyd algorithm. With the increasing of the number of vertices, the reduction of the number of cycles of our algorithm becomes more and more obviously.

Conclusion

In this paper, we improve the classical Floyd algorithm, which cannot effectively improve the search efficiency of the problem. In our algorithm in the directed graph and undirected graph, there are three conditions added to delete the vertex. The first vertex cannot be connected directly to the vertex. The second insert vertex cannot make the length of the path shorter. The third vertex weight is longer than the distance of the two vertices. The vertices are not involved in calculation, thus reducing the amount of calculation greatly. Therefore, our algorithm improves the search efficiency. The residence time is added at each vertex of the improved algorithm, which is more convenient to calculate the travel time. When the road cannot move, we need the *k-th* shortest path. Our algorithm is to remove some arcs and call the preparation algorithm to calculate the *k-th* shortest path. Our algorithms give the traveler multiple choice. In the future, we hope to find the *k-th* shortest path using much smaller time and faster running speed.

ACKNOWLEDGMENTS

This paper is granted by NSF of China (61572011) and NSF of Hebei province (A2013201119, A2014201033).

REFERENCES

Bondy JA and Murty USR (2008). *Graph Theory*, (San Francisco: Springer Press, California).

Dai XY and Cheng GZ (2013). Improvement and optimization of Floyd algorithm. *Journal of Xichang College* **26**(1) 63-65.

Eppstein D (1999). Finding the *k* shortest paths. *SIAM Journal on Computing* **28**(2) 652-673.

Li CJ (2006). A new algorithm to find the k shortest paths. *Journal of Shandong University* **41**(4) 41-42.

Wang SH (2009). *Graph Theory* (Beijing: Beijing Science and Technology Press, China).

Wang HY, Huang Q, Li CT and Chu BZ (2010). *Graph Theory Algorithm and its MATLAB Implementation*, (Beijing: Beijing Beihang University Press, China).

Xie Z and Li JP (1995). *Network Algorithm and Complexity Theory*, (Beijing: Beijing National University of Defense Technology Press, China).

Yan HB and Liu YC (2000). A new algorithm for finding short cut in a city's road net based on GIS technology Chinese. *Journal of Computers (in Chinese)* **23**(2) 210-215.

Zhang DQ and Wu GL (2009). Optimized Floyd Algorithm for Shortest Paths Problem. *Journal of Xuchang University* **28**(2) 10-13.