Copyright ⓒ2016 KSII

Distributed Optimal Path Generation Based on Delayed Routing in Smart Camera Networks

Yaying Zhang, Wangyan Lu and Yuanhui Sun
Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 201804, China
[e-mail: yaying.zhang@tongji.edu.cn, luwangyan91127@gmail.com, syhkiller@163.com]
*Corresponding author: Yaying Zhang

Received November 11, 2015; revised April 23, 2016; revised May 3, 2016; accepted May 19, 2016; published July 31, 2016

Abstract

With the rapid development of urban traffic system and fast increasing of vehicle numbers, the traditional centralized ways to generate the source-destination shortest path in terms of travel time(the optimal path) encounter several problems, such as high server pressure, low query efficiency, roads state without in-time updating. With the widespread use of smart cameras in the urban traffic and surveillance system, this paper maps the optimal path finding problem in the dynamic road network to the shortest routing problem in the smart camera networks. The proposed distributed optimal path generation algorithm employs the delay routing and caching mechanism. Real-time route update is also presented to adapt to the dynamic road network. The test result shows that this algorithm has advantages in both query time and query packet numbers.

Keywords: Distributed, delay routing, caching, optimal path

This research was supported by a research grant from the International S&T Cooperation Program (2012DFG11580) from Ministry of Science and Technology, China

http://dx.doi.org/10.3837/tiis.2016.07.013 **ISSN : 1976-7277**

1. Introduction

Nowadays vehicles number increases rapidly and continuously, road congestion and accidents have become the most common problems in urban traffic system. With the road network's spatial information, communicating system and real-time road state information that provided by Intelligent Traffic System(ITS) [1], the shortest path in terms of distance or travel time can be obtained between the vehicle's current location (source) and another location given by the driver (destination). Vehicles which can get more accurate path to the destination in time would effectively help reducing the urban traffic pressure.

Naturally, drivers tend to be more interested in the source-destination shortest path in terms of travel time, which we call *the optimal path*. Classical shortest path problems with fixed arc lengths have been studied intensively, resulting in the development of a number of efficient algorithms. There are many classical shortest path algorithms such as Dijkstra [2], A* [3] and also some special algorithms like Arc Flags [4], Precomputed Cluster Distances [5]. Because of their long query time or costly precomputation, these approaches could not recommend the optimal path when applied to the large dynamic road network. The shortest path information over a large dynamic road network in real-time would be highly desirable and beneficial. At present, the main method to deal with real-time traffic guidance in ITS is to gather the whole road network's information in data centers and generate the optimal path centrally.

As road traffic is inherently dynamic and time-dependent, finding the shortest path online in the time-dependent road network would be more challenging and difficult compared to the static road network. Variant of improved algorithms have been proposed. Dijkstra's algorithm is extended to the dynamic case through a recursion formula based on the assumption that the network has the FIFO property [6]. The A* algorithm with landmark (ALT) [7] used a lower bound of each edge during preprocessing. Kiseok Sung et al. [8] proposed a road network model with a time-dependent flow speed. A road network model based on history data had also be proposed by Gupta A et al [9]. Nevertheless, almost all of the above algorithms need to preprocess the history traffic data, which would be time-consuming and not be suitable in a large-scale road network with requirement of real-time performance. Furthermore the complication of time-dependent road network also makes these algorithms hard to be applied.

In the time-dependent road network, the actual travelling time of the optimal path is strongly related to the dynamic traffic condition on the road. A handful of studies have been focusing on improving the online computation of road segments' travelling time. Jinha Kim et al.[10] addressed a method to reflect the current road status by dynamically adjusting the speed patterns. However, it results in high processing cost during the query execution since the path updates need to be propagated to all the successive vehicles on that path. Hans-Peter Kriegel et al.[11] proposed two variants of a Lipschitz embedding to speed up common proximity queries such as distance range queries and k-nearest neighbor queries. A filter approach was employed to avoid unnecessary distance computations. But in time-dependent traffic network it still required much more memory and runtime than that of in the static network. In [12], Spyros Kontogiannis et al. provided approximate time-dependent distances in sublinear query time after preprocessing effort, and also gave the evaluation in experimental setting[13]. Nevertheless the high preprocessing effort and space consumption were impractical in use. The method named Customizable Route Planning in [14] offloaded most preprocessing effort to an initial metric-independent, separator-based phase which resulted in less query time. But as the weight changes rapidly in complexity the query time also raise higher. All these approaches either preprocess the history travel data or build a new network to

simplify the query process. We try to find a more suitable and simple approach to adapt to this time-dependent dynamic road network.

With the continuous deployment of video based traffic surveillance system, camera networks that capture data in public road networks are now in quite common use. In recent years, embedded smart camera with high-performance computing chips enables its applications in intelligent traffic monitoring and surveillance systems [15]. The on-site road cameras are mounted at road intersections. In addition to the task of traffic video capture, they can have some image processing work, e.g. event detection and vehicle identification as well as communicating with their neighboring cameras. The capability of smart cameras brings new solutions to the path finding problem in the dynamic road networks.

Motivated by the distributed spatial keyword querying in a dynamic road network proposed by Luo S. et al.[16] and the distributed graph query in the peer-to-peer network proposed by Srivatsa M et al.[17]. Both of them pose a novel approach to solve spatial keyword query or graph query in a distributed way, differing from the usual centralized setting. We apply the optimal path generation in a distributed way for better performance.

The major contribution of this paper is as follows: we build a real-time traffic flow model which maps the optimal path finding problem in the dynamic road network into the network routing problem in the distributed smart camera network. Furthermore, we employ the caching mechanism in the path routing. This improvement contributes to solve the problem of generating the optimal path in a large-scale concurrent query requests. It can speed up the query process and reduce the congestion of the data packages communicating in the distributed camera network. Besides, a route update algorithm is also proposed in this paper, which makes it more adaptive to the dynamic road traffic.

The rest of this paper is organized as follows: Section 2 states the problem description; Section 3 introduces the optimal path generation algorithm based on delayed routing and caching. Section 4 shows the simulation experiments and results analysis. Conclusions are given in Section 5.

2. Problem Description

We would state our problem and concerns for the development of our main contributions after clarifying some preliminary terms.

2.1 Preliminary Terms

Definition 1 (*Road Segment*): A road segment r is a directed edge that is associated with two road intersections ($r.s$, $r.e$) , $r.s$ and $r.e$ representing the start and end intersection of the road segment. Each road segment has a length $r.length$.

Definition 2 (*Road Network*): A road network is a weighted directed graph $G'(V',E')$, where V' is a set of vertices representing the road intersections of the road segments, and $E' \subseteq V' \times V'$ is a set of edges representing road segments. Every edge $e = < v_i, v_j > \in E'$ has a length weight $e.length$ indicating the distance from v_i to v_j.

Definition 3 (*Dynamic Road Network*): A dynamic road network is a road network defined in Definition 2 but with an additional time weight on every edge $e = < v_i, v_j > \in E'$. The time weight is represented by a time cost function $C_{vi,vj}(t)$, where t is the arriving time at the intersection v_i . $C_{vi,vj}(t)$ represents the travel time from v_i to v_j starting at time t.

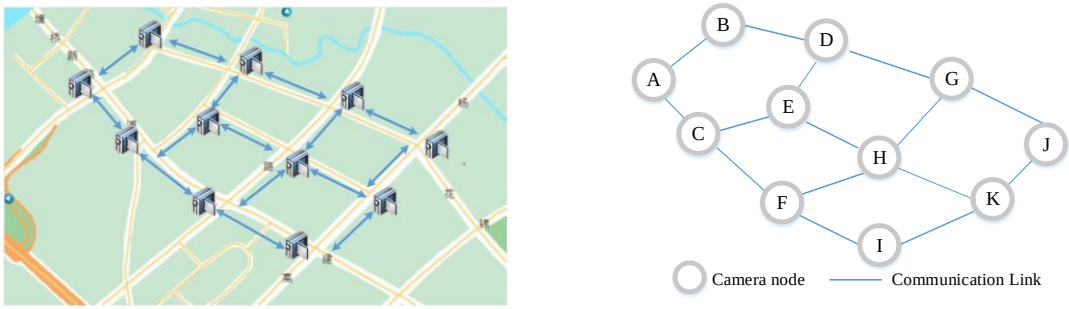
Definition 4 (*Path*): A path R is a set of connected road segments, i.e., R: $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_n$, where $r_{k+1}.s = r_k.e$; $(1 \leq k < n)$. The start point and end point of a route can be represented as $R.s = r_1.s$ and $R.e = r_n.e$.

Smart cameras are deployed at road intersections to serve traffic information collecting and processing. A driver would like to get the shortest path to a specific location during his/her driving journey.

Definition 5 (*Smart Camera Network*): A smart camera network is a undirected graph $G(V, E)$, where V is a set of vertices representing the cameras mounted either on the road intersections, and $E \subseteq V \times V$ is a set of edges representing communication link between neighboring cameras. Here '*neighboring*' means that there are road segments connecting with the adjacent intersections on which the cameras are mounted.

2.2 Problem Observations

The road network and camera network can be self-explained in **Fig. 1** It can be seen that the road network G' and the camera network G is isomorphic with their same topology. We assume that every smart camera mounted on the road intersection is capable of detecting the real-time traffic flow and vehicle information, e.g. the number of vehicles passing through, vehicle plate ID, color and type. Adjacent cameras can communicate with each other for information sharing and online collaboration. Thus queries on the road network G' can be mapped to operations in the smart camera networks G. However, the road network is inherently dynamic. The travel time on a road segment in the road network is varying over time. Therefore, how to find the optimal path (source-destination shortest path in terms of travelling time) in dynamic road network with the help of smart camera networks is our main concern.



(a) Road network G' with camera mounted (b) Smart camera network G

Fig. 1. Road network vs camera network

When dealing with mapping queries on the road network to operations in the smart camera network. The following should be concerned:

- A weighted directed graph fits the dynamic road network should be built. Every path query is launched in a built graph. The time weight of edges in the graph are the main factor influencing the result. Thus the time weight of edges must reflect the road traffic condition well.
- A suitable path query algorithm is needed. Many classic shortest path algorithms can be found to apply in some static weighted network. But algorithms that can adjust to the dynamic road network is highly desired. Furthermore, in the dynamic road network, when massive concurrent optimal path queries occur, the timeliness of the query results is also a big challenge.

- Dynamic route update. The vehicle's travel time on the road segment will vary by the time it travels through. To obtain the more accurate results, the time weight of edges of the built graph, which represents the condition of the road segments, should be updated in time.

We have the following assumptions in smart camera network G: (1) every camera as a vertex can communicate with its neighbors. (2) the message transmission time between cameras could be ignored comparing to the camera network routing delay time which indicates the vehicle travelling time through the road segment with cameras mounted. (3) the vehicles could be detected and uniquely identified by the camera using computer vision technology. Thus, the vehicle should be in one of the following two states: the vehicle is in the field of view of one camera v_i or the vehicle is on the road segment indicated by edge $<v_i,v_j>$, v_i and v_j are adjacent cameras.

We describe the optimal path generation problem as follow: when giving a pair of vertexes$^{v_s, v_d \in V'}$, try to find a path with a lowest sum of time weight from v_s to v_d currently.

The overview of path finding in a road network with smart cameras could be illustrated by the application scenario shown in **Fig. 1** When one vehicle made a path query request for the optimal path to a specific destination location (one road intersection or nearby) under the current road traffic condition, all cameras in the camera network will collaborate with each other to get the result. During the process of path querying, all cameras starting from the source location will broadcast a message which is just like dispatching a vehicle driving along the road. With the distributed smart camera network, concurrent attempts of different paths are made with the objective to find the optimal path.

3. A distributed optimal path generation algorithm

3.1 Real-time traffic flow modeling based on smart camera network

In road network with smart cameras, cameras are always monitoring vehicles passing by. When a camera detects a moving vehicle, it will transmit the vehicle's feature information to its neighboring cameras. The neighboring camera could identify the vehicles using the feature information it receives. Therefore, the vehicle's travel time between the two adjacent road intersections could be evaluated by the time difference of its presence in the neighboring smart cameras mounted on the respective adjacent road intersections. This is feasible under the assumption that smart camera can detect and identify vehicles. Thus, we can estimate and predict the average travel time between two adjacent road intersections by collecting the travel time of a certain amount of vehicles passing along this road segment.

Iterative learning control (ILC)[18] has been successfully applied to traffic flow estimation for a macroscopic freeway environment. However, due to the absence of the vehicle's travel time, traffic parameters such as traffic density, space mean speed, the length of road segments and sample time intervals have to be considered to get the traffic flow leaving and entering the road intersection with iterative learning[19]. These parameters are essential to get the travel time between the two adjacent road intersections more accurately. With the assumption that a vehicle's feature could be obtained by the smart cameras and that feature could be transmitted to the neighboring cameras. The neighboring cameras can get the travel time by capturing the same vehicle. The average travel time can be obtained through amounts of the vehicle's travel time. In this means, we could simplify the iterative learning control approach to iteratively learn the average travel time between two adjacent road intersections by cameras' capturing of passing vehicles.

Assume the average travel time previously is T_{old} and the average travel time calculated by the smart cameras in the current minute is T_{new}. The current average travel time T_{cur} can be learned by T_{new} and T_{old} as in equation (1).

$$T_{\text{cur}} = \beta T_{\text{old}} + (1-\beta) T_{\text{new}} \tag{1}$$

Where β is a constant coefficient, $\beta \in (0,1)$

The iterative learning process is as follow steps:

(1) T_{old} is initialized to zero;
(2) Obtain the T_{new} from the smart camera network;
(3) get T_{cur} by equation (1);
(4) set $T_{old} = T_{cur}$;
(5) at next minute, repeat steps (2)-(4);

Thus the vehicle's travel time on road segments could be obtained by the above iterative learning at a certain interval i.e. one minute.

3.2 Routing Delay Time vs Vehicle Travel Time

Suppose the vehicle travel time from road intersection A to road intersection B under the current traffic condition is T minutes, then we can have t milliseconds from smart camera C_A to wait before it broadcasts the message to smart camera C_B. In order to ensure that the shortest routing on the smart camera network can be used to represent the shortest driving-time path on the road network, we have the routing delay time t (in milliseconds) be linear to the vehicle travel time T (in minutes) as define in equation (2)[20].

$$t = k \times T \tag{2}$$

Where k is transforming coefficient.

We need this time transformation due to the following observations:

- Routing delay is directly proportional to driving time. This means that the original road network with weight of road segments could be transformed into an isomorphic network with weights scaling down. Thus the shortest routing path on the smart camera network would be the shortest path on the original road network.
- The time for one camera to broadcast a message to its neighboring cameras is very short and almost the same (i.e. 0.2ms). However, in the real road network, the time for vehicle to travel to different adjacent road intersection varies because the length of the adjacent road segments are different and traffic condition are always changing over time. Thus, in order to fulfill the transformation from optimal path in road network to shortest networking routing in smart camera network, delay is needed before the broadcast of every routing message to indicate the traffic condition on the real road. This delay time for routing is determined by the vehicle travel time between the adjacent road intersections under its current traffic condition.

The delay time is selected varying from 10ms to 1000ms so as to avoid the interference of data processing with message broadcasting. The value of k in equation (2) is important in affecting the accuracy and real-time performance of the shortest path query. A smaller value of k will lead the message broadcast delay time be close to the message transmission delay between camera nodes, which will cause inaccurate result. A bigger value of k will have longer waiting time before message broadcasting, which would result in a longer query time. In this paper, our experiments show that a value of 10 is suitable for k to get a good performance on both accuracy and efficiency.

3.3 The distributed path finding algorithm based on delayed routing

When delay time is introduced in the smart camera network, the roads′ condition or the travel time on a road segment is represented by the transmission delay between cameras on adjacent road intersections, which associates the distributed camera network with the real dynamic road network.

After building the weighted directed graph of the smart camera network, every smart camera node will maintain *neighbor table* and *query message table* to guarantee forwarding messages in a right way. The structure definition of the tables and format of *query packet* and *response packet* is in **Table 1**.

The neighbor tables are mainly to maintain the transmission delay of adjacent nodes which represent the weight of edge in graph. The query message tables record the passing packets, and drop the repeated packets by their unique identifier (i.e. GUID), that can be helpful to reduce the numbers of redundancy packets. Furthermore, the query message tables also guarantee the correctly backtrack from destination node and record the optimal path. For example, Transmission delay information of camera *A* is shown in **Table 2** for the smart camera network in **Fig. 2**.

Table 1. Structure definition and Packet format

Name	Definition
query packet	<Type,GUID, DstID, SrcID, StartTime>
response packet	<Type,GUID,DstID,SrcID,Path,startTime>
query message table	<GUID, LastID, Cost>
updating packet	<Type, Start, End, NewCost>
neighbor table	<NeighborID, Cost>
route table	<DstID, LastID, Cost>
Type	packet's type
GUID	packet's ID, globally unique
DstID	destination node's ID
SrcID	source node's ID
LastID	last hop node's ID
NeighborID	neighbor node's ID
StartTime	query start timestamp
Path	the passing route of packet
Cost	the weight of road segment

In the real application, the branch of a road intersection will always vary from 3 to 6. Thus, the number of neighboring cameras normally differs from 3 to 6 and a linear list can be used to organize this neighbor table for query and update.

Table 2. the neighbor table of Camera node *A*

Neighboring camera ID	Transmission delay
D	36ms
B	20ms
…	….

As in **Fig. 2**, when camera *A* receives a path query request to the location where camera *S* mounted. It will generate a query packet and send it to its neighbors *B*, *D* with their delay time. When the intermediate cameras receive the query packet, they will also transmit it to their neighbors with their own delay time. For example, camera *B* will send this message to camera *C* and *E* with a delay time of 19ms and 32ms respectively. So the query packets transmit in the road network act like vehicles driving on the road network. Eventually, the destination camera *S* may *receive more* than one query packet. But only the first arriving one contains the optimal path. Camera *S* will generate a response packet with the optimal path and send back to the source camera *A*. The detailed query and response process will be described in Section 3.4.

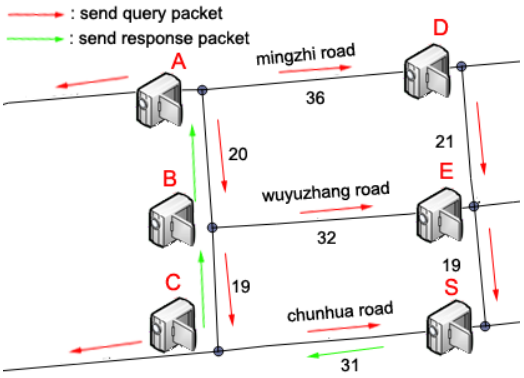


Fig. 2. The smart camera network with transmission delay

3.4 Path routing caching

For every path query request, the cameras should always do the querying and computing to get the result. Massive and concurrent query request would cause redundant message broadcasting and computations in a large scale dynamic environment. Thus, we introduce the idea of route table in the computer network to the path finding algorithm.

We find that it is possible to get the shortest-path-tree[21]. In this shortest-path-tree, the path between arbitrary parent node and child node is the shortest path, and the path between arbitrary child node and parent node is also the shortest path. So we could combine the well-known routing mechanism of computer network with the distributed path query algorithm. Every camera in the road network will maintain an extra route table which stores the shortest-path-tree. And an updating packet is also defined aiming to adapt to the dynamic road network.

Each smart camera would maintain a route table and a neighbor table. For every query request, the camera node will check if the local route table has already had the result to the requested destination; if not, it will start the delay routing algorithm to find the optimal path. During this process, each node related to the query will check their route tables to reduce the query time. When the destination node send back a response message, the node receiving the message will also store and update the optimal path tree in the passing nodes, which reduces repeated computations. To keep the road network dynamic and get the latest query result, the nodes will start the route update algorithm to refresh nearby nodes' route table after the weight of some road segment changes beyond a certain threshold to reflect the latest real-time road traffic condition.

A route table is a collection of triples: <*DstID, LastID, Cost*>. Updating packet has the format as <*Type, Start, End, NewCost*>, where *Start* means the source of changed road segment and *End* means the destination, the *NewCost* means the new weight indicating the

new traffic condition on that road segment..

Route table as defined can cache several shortest-path-tree with itself as the parent node. For example, in the distributed camera network shows in **Fig. 3**, camera node *S* will maintain a *neighbor table* as the **Table. 3**. When the vehicle request for the path from *S* to *T*, we can get the path of first packet arriving at *T* is *S-C-F-E-T*. After this query, we can get node *C*'s query message table like **Table 4** and route table like **Table 5**.

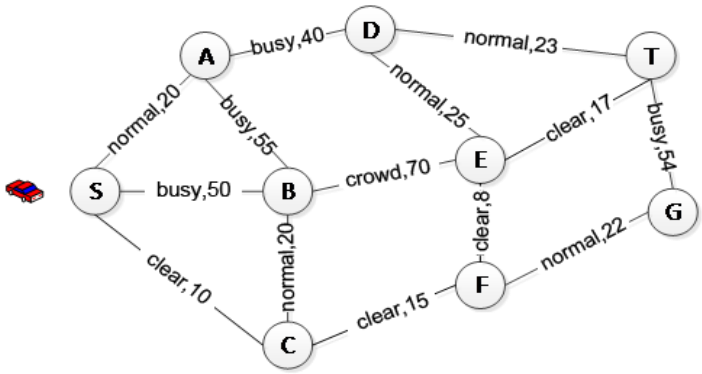


Fig. 3. A distributed camera network

Table 3. Node S's neighbor table

Node ID	Delay/ms
A	20
B	50
C	10

Table 4. Node C's query message table

GUID	Last hop node	Delay/ms
123456	S	10

Table 5. Node C's route table

Destination Node	Last hop node	Delay/ms
F	C	15
E	F	24
T	E	17

The process of the distributed path query based on delay routing and caching is shown in **Algorithm 1** for source node and **Fig. 4** for neighbor node.

Algorithm 1 SourceNode Receive

Input: P, the received packet; RT, router table
Output: path, the optimal path

```
1: if  p is RequestPacket then
2:      if RT[p[dstid]] not null then
3:          return getPath(p)  // get the optimal path from route table
4:      else // route table does not contain any related path
5:          send p to neighboring nodes
6: else  // p is ResponsePacket
7:      receiveReplies(p)  // receive ResponsePackets and deal with them
```

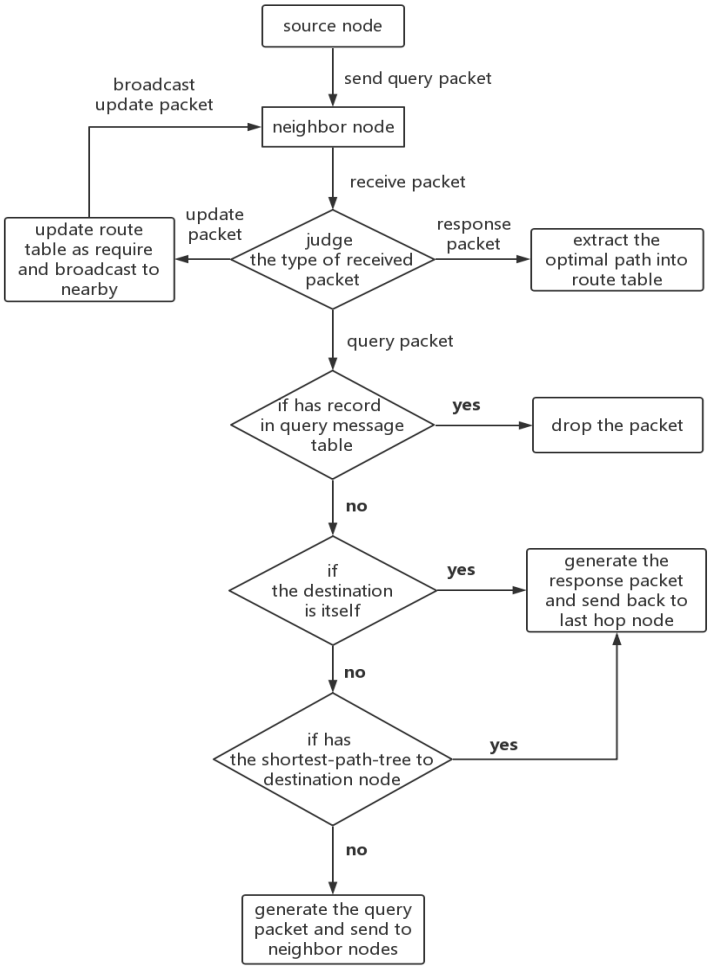


Fig. 4. The query process of neighbor node

When a path query request is sent to a node in the distributed camera network,

(1) The source node will first check if the shortest-path-tree to the destination is contained in its route table. If it exists, which indicates this path have be queried before and already be cached in its route table, the path in the shortest-path-tree would be returned as the optimal path. Otherwise, it will broadcast this query request to its neighbors in terms of its neighbor table. For example, as the road network shows in **Fig. 3**, if the node *S* receive a query request to destination *T* and also there is no cache to the destination in its route table, it will generate a query packet. With its neighbor table, it will send this query packet to node *C* after a delay time of 10ms, and 20ms to *A*, and 50ms to *B* respectively.

(2) When the neighbor node receives a packet, it will first examine the type of this packet. If it is a query packet, it will check the query message table. If there is a record with the same GUID in the query message table, this packet will be dropped. The same GUID means that this packet have arrived before and no long needs to be processed, which would eliminate the chance for the repeated packet sending. If the destination of this packet is just the node itself, the node will generate a response packet and send back through its previous passing nodes. If not, it will finally check its route table to find if there is a record to the destination. If no record

for this query exists, it will start broadcasting to its neighbors. If the packet is a response packet, the node obtains shortest-path-tree from the path field of this packet and stores it in its route table, then sends back the response packet towards the source node.

(3) Finally, the source node will have three situations to deal with before getting the exact result due to the caching mechanism. The first one is finding the shortest-path-tree in its local route table directly. The second one is that the source node receives several response packets, which includes a path part from real-time query and another part from the route table, or a path that totally from the real-time query. The third one is the mixture of real-time query path and the shortest-path-tree from route table.

To handle the last two situations, **Algorithm 2** is presented in pseudo code. After the source node receives the response packets, it calculates each packet's total time weight T by its returned optimal path. Then each packet's query time R will be obtained by its *StartTime* field and the received time. We need to compare T and R to determine which packet contains the optimal path. If $T > R$, it means the response packet's optimal path contains cached path and query path. So the node will start a thread to wait a period time of T-R and then return its path as the optimal path. During this period, any packets arrive to this node would also do that comparison and wait. Finally, all threads will be stopped, any other arrived packets will be dropped and the optimal path will be obtained either from a path with cached path or a newly-got query path.

Algorithm 2 Receive Replies

```
Input: p, the received packet
Output: path, the optimal path
1: if this[id] == p[dstid] then
2:     finalPath ← getPath(p) // get optimal path from route table
3:     insertCache(finalPath) // insert the short-path-tree into route table
4:     T ← getPathTime(p) // get the total time weight from the a returned path
5:     R ← NOW - p[startTime] // get the query time
6:     if T > R then // response packet's optimal path contains cached path and query path
7:         schedule(T- R) // produce a timer to take a wait
8:     else
9:         schedule(0) // directly get the path as the optimal path
10: else
11:    sendBack(p) // send the response packet to next processing node
```

This algorithm ensures the accuracy of the query results, and the query time will be no more than a totally non-cached path query time with the caching mechanism. The best situation is getting the result from local route table directly. When there are lots of query request, the shortest-path-tree cached in the node will increase which will improve the cache hit ratio. This algorithm is much more efficient in the large-scale network with concurrent requests.

3.5 Route updating

In the dynamic road network, the time weight of the edges represents the traffic condition of a road segment which is inherently changing over time. Thus, the router table of camera nodes must update in time to guarantee that vehicles can get the latest and more accurate query result.

The route table of the camera nodes will be updated after the weight of some road traffic condition changed beyond a certain threshold. For example, as the road segment *SC* shown in **Fig. 3**. If it is the one-way road from *S* to *C* and when the weight of road *SC* changes beyond

the threshold, the node S will broadcast the update packet to the nearby nodes within a certain range. The node receives the packet will check its route table. If it contains the *S*-*C* (from *S* to *C*) subtree, then update its *Cost* field. Otherwise, the packet is dropped. If the road segment *SC* is a two-way road, the node *S* sends the broadcast packet to update *S*-*C* subtree, and the node *C* will send broadcast packet to update *C*-*S* subtree.

The pseudo code of the route update algorithm is in **Algorithm 3**:

Algorithm 3 Update Route

Input: p, start, end, delay

1: **if** --*TTL* > 0 and ∃ *end* ∈ routeTable and routeTable[*end*] == start **then**
2: changeDelay(*start*,*end*) // update the route table when the weight changed beyond threshold
3: p ← createBroad(*start*,*end*,*delay*) // create a broadcast packet to update the neighbors' route tables
4: broadCast(*p*)

3.6 The limit of broadcast area by elliptical model

The above path query algorithm has reduced the redundant query and the broadcast packets in the network through the cache on route table. However, every query will cause the neighbors broadcast their packets without any limitation, which still produces tremendous redundant packets in the network and influences the camera nodes' work efficiency. Actually, the optimal path is normally always around the straight line path from the source to the destination and the camera nodes in the real road network distribute uniformly. Thus, we can use the elliptical model [22][23] to limit the message broadcasting in a certain area.

In the distributed smart camera network, we store the longitude and latitude of the camera's location in every node to determine its location. When the node *S* start a query request to node *T* in the network shown in **Fig. 3**.The node *S* will get the coordinate of destination node, and then build an ellipse like in **Fig. 5** by considering *S* and *T* as the focuses of the ellipse and a coefficient of their straight distance as the major axis.

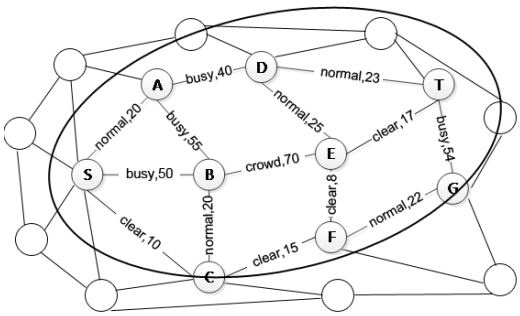


Fig. 5. The elliptical area

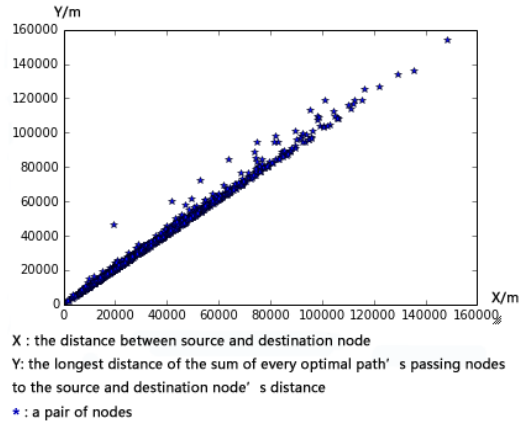


Fig. 6. The result of intermediate calculating about the elliptical coefficient

We get the proportional coefficient between major axis and minor axis by statistical analysis. The main process is as follows: Taking Beijing's city map as an example, we randomly selected 2000 pairs of source and destination nodes. Then we calculated the

straight-line distance between the node pairs and the longest distance of the sum of every optimal path's passing nodes to the source and destination node's distance. The **Fig. 6** shows the result of our test. As the node pairs' straight-line distance increases, distance related to the elliptical coefficient also raise steadily. We can find the points in the figure almost cluster in a straight line, which means a city's elliptical model can nearly be built only by the elliptical coefficients. Then the Linear Least Squares is used to fit a line and the slope of the line is considered as the elliptical coefficient.

With the elliptical model, the node needs to check if the target node is in the elliptical area. If not, it would ignore this broadcast message, which can reduce many redundant packets in the network and improves the efficiency of nodes further more.

4. Simulation and Result Analysis

We test the performance of this algorithm with large-scale concurrent requests. We do the simulation on the NS2 [24] with part of the road network of Beijing containing about 1283 intersections and 1631 road segments. We define the weight of road segment by a function of the road's distance and a random traffic condition. And we did the following test:

- In the situation of large-scale concurrent requests, comparing the cumulative query time and the total numbers of sending packets of the algorithm with and without caching mechanism.
- Changing the weight of a road segment beyond the threshold, and comparing the result before and after the route updating.
- Observer the effect after applying the elliptical model, reducing the number of nodes taking part in query broadcasting in the network.

To make the nodes generate much more caches in a short time, we choose the node pairs of same source node and random destination nodes. Then we start a large numbers of requests in 10s and record the cumulative query time and broadcast packets. Finally, we get the result shown in **Fig. 7** and **Fig. 8**.

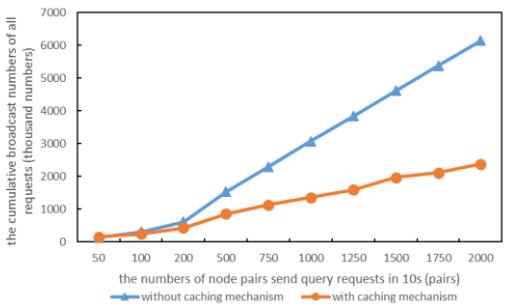


Fig. 7. Cumulative broadcast packets

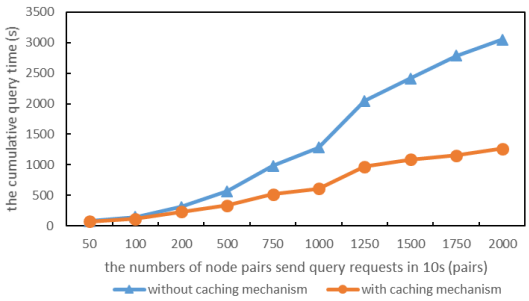


Fig. 8. Cumulative query time

We find that with the increasing number of requests from 50 pairs to 2000 pairs, both the cumulative broadcast packets and query time of the algorithm with caching mechanism increase more slowly and performs more stable than that of the algorithm without caching mechanism. In the beginning, before the query requests increase to 500, the two approaches have almost the same result. With the query requests increasing, the cached approach keeps

increasing slowly. However, the none-cached approach need nearly 1.5 sec for per query request at last which is three time longer than that of cached approach in query time. This is because the large numbers of requests make the nodes' route tables caching more and more shortest-path-tree and reducing the repeated queries. It is observed that if there is little queries in the road network, the cache in the nodes' route tables will decrease, and the algorithm's efficiency will be limited.

In the second test, we test the route updating algorithm. We firstly get the optimal path by the path query algorithm. Then we change one of the road segments' weight beyond the threshold, which may simulate a traffic accident. After that, starting the path query again to get the newest optimal path. The result is shown in **Fig. 9**. At first, the path A-B-C-G from S to T is the optimal path in that map. When the road segment's weight is dynamically changed beyond the threshold as 20ms(in this experiment), in Fig.10 we manually change the weight of road segment A to 50ms, that increase 27ms which beyond the threshold we set up before. Then the route update algorithm will be activated. It sends the broadcast packet with updated message to the nodes within a certain range. And the nodes that receive the updated packet will check their route table and updated the changed cache. So after changing road segment A's weight, the weight of A+B+C is bigger than the weight of D+E+F. The result is shown in **Fig. 10**. The line in red represents the optimal path. When some road segments' weight change frequently, the route update algorithm might become practically inefficient.

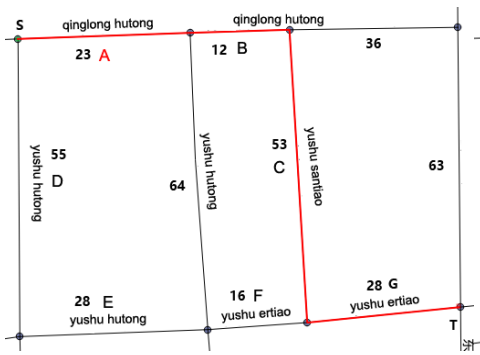


Fig. 9. The optimal path before updating

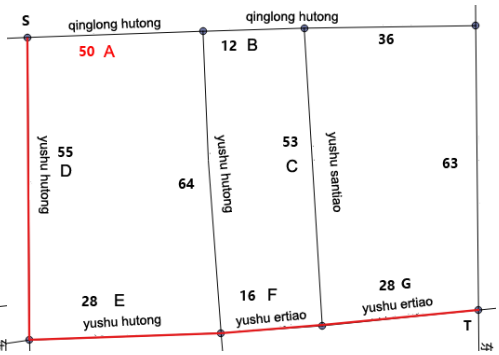


Fig. 10. The optimal path after updating

In the third test, we compare the number of broadcast nodes with and without elliptical limit. We choose 10 pairs of nodes in the Beijing road network. With the increasing distance between the nodes, the numbers of nodes that the query packets have passed in using elliptical model and without elliptical model are in **Fig. 11**. After increasing the distance between source and destination from nearly 10000 to 90000 meters, the elliptical model's limitation controls the query packets in a very steady numbers compared to the none-limit situation which increases rapidly and almost eight times than using an elliptical limit at the last query pairs. Using the elliptical model effectively limits the area of nodes' broadcasting and much more redundant packets will be dropped before broadcasting.

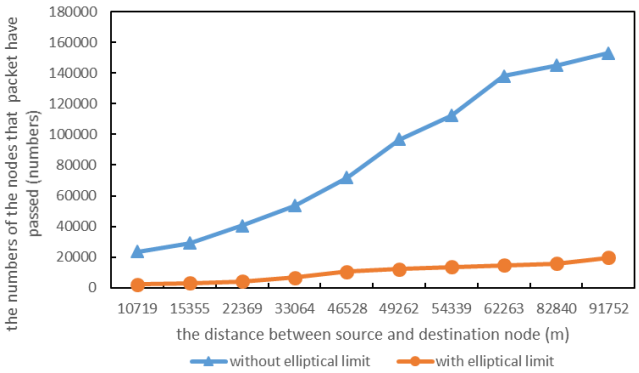


Fig. 11. The message broadcasting numbers comparison

Finally, we take abundant query requests between the nodes either nearby or far away in the city of Beijing's road network into test. We found each query request between nodes involves a certain range of the road network which only affected by the distance between the nodes. Moreover, after applying the caching mechanism and elliptical limit, every query only produces thousands of packets in the network which also contain TTL field to drop themselves. So we consider the proposed algorithm feasible when applied in practice.

5. Conclusion

In this paper, we proposed the distributed path query algorithm based on delay routing. We use caching mechanism to reduce the redundant queries in the large-scale concurrent requests situation. During the query process, the nodes cache the shortest-path-tree extracting from the response packets in their route tables. Furthermore, when one of the road segment's weight beyond the threshold, the node will start the route updating algorithm to refresh the route table of the node within a certain range, which guarantees that the query result can be more optimal and accurate. Besides, we apply the elliptical model in this algorithm to reduce the numbers of nodes that the query packets have passed along. Eventually, the simulation result shows that this algorithm can effectively reduce the query time and the numbers of broadcast packets.

References

[1] Bishop R., "A survey of intelligent vehicle applications worldwide," in *Proc. of Intelligent Vehicles Symposium, 2000. IV 2000. the IEEE. IEEE*, 25-30, 2000. Article (CrossRef Link)

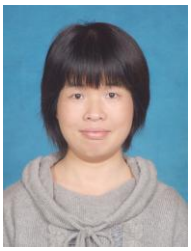
[2] D.K. Fan, "P. Shi. Improvement of Dijkstra's algorithm and its application in route planning," in *Proc. of 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, IEEE, 1901 – 1904, 2000. Article (CrossRef Link)

[3] D.S. Yershov, S.M. Lavalle, "Simplicial Dijkstra and A * Algorithms for Optimal Feedback Planning," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots & Systems*, 32(14):3862 – 3867, 2011. Article (CrossRef Link)

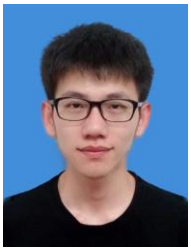
[4] U. Lauther, "An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags," in *Proc. of 9th DIMACS Implementation Challenge*, 19-39, 2006. Article (CrossRef Link)

[5] J. Maue, Sanders P, D. Matijevic, "Goal Directed Shortest Path Queries Using Precomputed Cluster Distances[M]," *Experimental Algorithms*, 316--328, 2006. Article (CrossRef Link)

[6] Y.B. Huang. "An improved dijkstra algorithm for the graphs of Shortest-Path tree," *Journal of Beijing Institute of Machinery*, 2002. 17(4):50-55. Article (CrossRef Link)
[7] V. Andrew, "Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory," in *Proc. of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005. Article (CrossRef Link)
[8] S. Kiseok, M.G. Bell, M. Seong, S. Park, "Shortest paths in a network with time dependent flow speeds," *European Journal of Operational Research*, 121(1):32-39, 2000. Article (CrossRef Link)
[9] Gupta A, Lakshmi J, Nandy S K., "Real Time Routing in Road Networks," in *Proc. of Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on. IEEE*, 9-16, 2014. Article (CrossRef Link)
[10] Kim J, Han W S, Oh J, et al., "Processing time-dependent shortest path queries without pre-computed speed information on road networks," *Information sciences*, 255: 135-154, 2014. Article (CrossRef Link)
[11] Kriegel H P, Kröger P, Renz M, et al., "Proximity queries in time-dependent traffic networks using graph embeddings," in *Proc. of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*. ACM, 45-54, 2011. Article (CrossRef Link)
[12] Kontogiannis S, Wagner D, Zaroliagis C., "Hierarchical oracles for time-dependent networks," 2015. Article (CrossRef Link)
[13] Kontogiannis S C, Michalopoulos G, Papastavrou G, et al., "Analysis and Experimental Evaluation of Time-Dependent Distance Oracles," *ALENEX* 147-158, 2015. Article (CrossRef Link)
[14] Delling D, Goldberg A V, Pajor T, et al., "Customizable route planning in road networks[J]," *Transportation Science*, 2015. Article (CrossRef Link)
[15] B. Heyman, M. Paindavoine, R. Schmit, L. Letellier, T. Collette, "Smart camera design for intensive embedded computing," *Real-Time Imaging*, 11(4):282-289, 2005. Article (CrossRef Link)
[16] Luo S, Luo Y, Zhou S, et al., "DISKs: a system for distributed spatial group keyword search on road networks," in *Proc. of the Vldb Endowment*, 5(12):1966-1969, 2012. Article (CrossRef Link)
[17] Srivatsa M, Kawadia V, Yang S., "Distributed graph query processing in dynamic networks," in *Proc. of Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on. IEEE*, 20-25, 2012. Article (CrossRef Link)
[18] Owens D H, Freeman C T, Dinh Van T., "Norm optimal iterative learning control with intermediate point weighting: theory, algorithms and experimental evaluation," *IEEE Transactions on Control Systems Technology*, 21(3): 999-1007, 2013. Article (CrossRef Link)
[19] Z.S. Hou, S.T. Jin, M. Zhao, "Iterative learning identification method for the parameters of the macro traffic flow model," *Automation Journal*, 34(1):64-71.(in Chinese), 2008. Article (CrossRef Link)
[20] Q. Long, C. Ye, Y.Y. Zhang, "The distributed path generation algorithm based on real-time road condition in the dynamic road network," *Computer Science*, 41(9):259-262,278.(in Chinese), 2014. Article (CrossRef Link)
[21] J. Tang, D.L. Zhang, "Fast path generation algorithm based on routing mechanism in the variable weight network," *Computer Science*, 38(12):110-112.(in Chinese), 2011. Article (CrossRef Link)
[22] S Nordbeck, B Rystedt, "Computer cartography:shortest route program," *The Royal University of Lund*. 1969.
[23] Luo P, Qiu Q, Zhou W, et al., "A Shortest Path Algorithm Suitable for Navigation Software," *Computer Engineering and Networking. LNEE 277*, Springer International Publishing, 153-161, 2014. Article (CrossRef Link)
[24] The VINT Project, The NS Manual. Article (CrossRef Link)



Yaying Zhang is an Associate Professor with the Key Laboratory of Embedded System and Service Computing, Tongji University, Shanghai China. She received the B.S. degree in computer science and the M.S. degree in electrical engineering from Shandong University of Science and Technology, Shandong, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from Shanghai Jiaotong University, Shanghai, China, in 2004. Her research interests include distributed computing and intelligent traffic systems.



Wangyan Lu is an M.E. candidate in computer science of Tongji University, Shanghai, China. He received the B.E. degree in network engineering from Zhejiang University of Technology in 2014. His research interest is route planning.

Yuanhui Sun received the M.E. degree in 2016 from Tongji University, Shanghai, China. He received the B.E. degree in Software engineering from Zhejiang University of Technology in 2013. His research interests include path restore and route planning.