K Nearest Neighbor Path Queries based on Road Networks

Lulin Chen[1], Guofeng Qin*[2]
Department of Computer Science and Technology, Tongji University, 4800 Caoan Road, Jiading District, Shanghai, China
*Corresponding author, e-mail: lulinchen1989@163.com[1], gfqing@yahoo.com.cn[2]

Abstract

The Island is a k nearest neighbor query algorithm of moving objects based on road networks and can effectively balance the performance of query and update. But the algorithm doesn't consider the direction of moving object which is required in many scenarios. It traverses vertexes from all directions, meaning wasting a lot of time in visiting useless vertexes. Besides, it doesn't return query path. In this paper, we propose an improved Island algorithm which takes full account of moving direction. It takes best point estimate and heuristic search method, effectively reducing the number of traversal vertexes. At the same time, we optimize the data structure and let it return the query path. Results show that the improved algorithm outperforms the original one. Finally, we describe electric vehicle charging guidance system based on the improved Island algorithm.

Keywords: *nearest neighbor query, island algorithm, heuristic search*

1. Introduction

Due to the advances in mobile communication and geographic information technology, location-based services are increasingly popular [1]. The k nearest neighbor question is an important class of query type [2] among location-based services. It is used to find the nearest k objects from query object, such as “find nearest five restaurant's positions for taxi” [3], “find nearest k gas station's positions for drivers” [4]. There are some query algorithms for static objects based on road networks, like Papadias's INE (Incremental Network Expansion) algorithm [5], Kolahdouzan's VN3 (Voronoi-Based Network Nearest Neighbor) algorithm [6] and Huang's Island algorithm. By offering flexible yet simple means of balancing re-computation and pre-computation, the Island algorithm is able to manage the trade-off between query and update and has better performance than other two methods [7]. But the algorithm's online network-expansion component much like Dijkstra algorithm. The main characteristic is the query object as the center outward expansion layers, until k objects extended. Although the algorithm can get k shortest distance objects, it doesn't consider the direction of moving object which is required in many scenarios. What's more, it only gives nearest objects' position but not return query paths. This article describes an improved Island algorithm, and it optimizes the algorithm from network-expansion component and data structure. By using best point estimate and heuristic search method, the algorithm reduces the number of traversal vertexes to increase query speed, meanwhile it gives query path which can be used in path navigation.

2. Road Network Model

In many practical applications, such as map resource services, road navigation system, the road network is usually represented as a collection of a large number of nodes and road segments. Road nodes can be divided into the following three cases: (1) the intersections of the network, (2) the dead end of a road segment, and (3) the points where the curvature of the road segment exceeds a certain threshold so that the road segment is split into two to preserve the curvature property [8]. Taking the one-way street into account, we can transform the network into a directed graph in memory. Figure 1 shows an example of road network and its corresponding directed graph.

Received April 21, 2013; Revised June 22, 2013; Accepted July 23, 2013

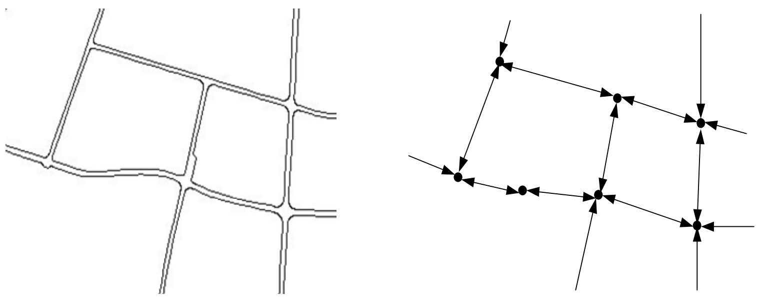


Figure 1. Road Network and its Directed Graph

There are some definitions for graph as follows:

Definition 1: A road network is a directional weighted graph $G = (V, E)$. It consists of a set of vertices *V* and a set of edges *E*, where $E \subseteq V \times V$. Each edge can be represented as $e_{i,j}=(v_i,v_j,l)$, $e_{i,j} \in E$,$i \neq j$, where v_i and v_j are the starting and ending vertex, *l* is the length of the edge.

Definition 2: Using *query point (qp)* to denote a moving object and *data point (dp)* to denote a static query object.

Definition 3: A location on the road network can be denoted as *loc = (e, pos)*, where *e* is the edge on which the location is located and *pos* represents the distance from the starting vertex of the edge to *loc*.

We have two conditions: (1) *qp* such as cars run on a road network and *dp* such as gas stations are located on the road network. (2) The distance measure is defined as the shortest path length (network distance) on the network [9]. In Figure 2, $e_{1,7}=(v_1,v_7,5)$, *qp*'s $loc=\{(e_{4,5},1), (e_{5,4},3)\}$ and dp_1's $loc = \{(e_{1,7},3), (e_{7,1},2)\}$.

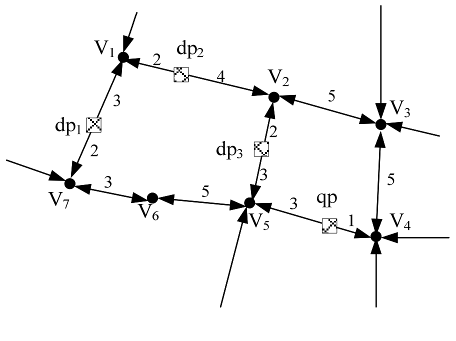


Figure 2. Query Point and Data Point in the Directed Graph

3. Island Algorithm

The Island algorithm consists of pre-computation component and an online network expansion component. Pre-computation component divides the network into some areas, we call them island. Each island is a circle taking *dp* as the center and a value given in advance as the radius. In order to record the island information, every vertex stores references to the islands that cover the vertex and the network distances from the vertex to the data points. When getting a query requirement, we first need to check the islands that the *qp* is inside and maintains the *dp* found in a priority queue, then starts a network expansion to find borders of new islands. The expansion process terminates when the priority queue has k elements [7].

Q_{dp} and Q_v are priority queue for recording intermediate data. Q_{dp} is used to record the covered data points and their distances to *qp*. The element of Q_{dp} is denoted as $(dp_k, d(qp,dp_k))$ and its size is limited to k. Similarly, Q_v records the candidate points and their distances to *qp*, using $(v_x, d(qp,v_x))$ denotes elements. Both queues sort elements by the distance and don't allow duplicate data. The algorithm can be described as follows:

(1) For each *dp* on edge *qp.e*, put *(dp, d(qp,dp))* into Q_{dp} and put *(qp.e.*v_s*, d(qp, qp.e.*v_s*)*, *(qp.e.*v_e*, d(qp, qp.e.*v_e*)* into Q_v;
(2) For each *dp*, if its island covers *qp.e.*v_s or *qp.e.*v_e ,put *(dp, d(qp,dp))* into Q_v;
(3) If the size of is k, the query terminates, else start network expansion;
(4) Get Q_v's first element *(v, d(qp,v))* and mark it visited;
(5) For each non-visited adjacent vertex v_x of *v*, put *(*v_x*, d(qp,*v_x*))* into Q_v when Q_v doesn't contain v_x. If v_x has already existed in Q_v and *d(qp,* v_x*)* is smaller than the distance stored, update the distance to *d(qp,* v_x*)*, else do nothing;
(6) Repeat steps 4 and 5 until finding k data points.

4. The Improved Island Algorithm

Island algorithm's expansion process is similar to Dijkstra's single source shortest paths algorithm. Despite the high accuracy, the algorithm doesn't consider the specific circumstances of the query points and data points, which means wasting a lot of time in visiting useless vertexes. So the search is blind and inefficient. We can use the best point estimate and heuristic search method to cut down the traversal vertexes and speed up the query.

4.1. Algorithm Description

Consider the following two factors: (1) In daily life, when making path planning and navigation, if you want to go south will never go north, this is the habitual choice; (2) A straight line between two points is the shortest. The shortest path between two points on the actual road network must be closer to the line. If the straight line between one *dp* and *qp* is the shortest, it represents the *dp* may also be the nearest one to the *qp* on the road network. So we can use the function to evaluate data points around the *qp*:

$$E(dp) = A(dp) * L(dp), A(dp) < 90$$

In the formula, *A(dp)* represents the angle between the direction of *qp*'s movement and the direction from *qp* to *dp*. The smaller the value of *E(dp)* which means the better the *dp* that match the query expectations. And then select the best k data points, but these data points are not the final result, they just represent the forward direction of the search.

After determining the best data points, we can use the heuristic strategy for searching. It limits the search scale by an evaluation function. In each step of searching process, we find the vertex with the lowest value of the evaluation function as the next extension vertex. Here, the evaluation function is:

$$f(v_x) = g(v_x) + h(v_x), h(v_x) = L(Vdp)$$
$$Vdp.Lat = \sum_{i=1}^{K} dp_i.Lat, Vdp.Lon = \sum_{i=1}^{K} dp_i.Lon$$

In the formula, $g(v_x)$ is actual cost from *qp* to v_x and $h(v_x)$ is the straight line distance between v_x and virtual data point. Virtual data point's longitude is the average of k best data points' longitudes. Similarly, virtual data point's latitude is the average of k best data points' latitudes. The algorithm can be described as follows:
(1) For each *dp* on edge *qp.e*, put *(dp, d(qp,dp))* which is in the moving direction into Q_{dp} and put *(qp.e.*v_e*, d(qp, qp.e.*v_e*)* into Q_v;
(2) For each *dp*, if its island covers *qp.e.*v_e ,put *(dp, d(qp,dp))* into Q_v;
(3) If the size of is k, the query terminates, else start network expansion;
(4) Assuming we have found n data points, there are still m (m=k-n) data points needed. Use best point estimate method to find other m optimal data points and to calculate the virtual data point.
(5) Get Q_v's first element *(v, d(qp,v))* whose *f(v)* value is the smallest , and then mark it visited;
(6) For each non-visited adjacent vertex v_x of *v*, put *(*v_x*, d(qp,*v_x*))* into Q_v when Q_v doesn't contain v_x. If v_x has already existed in Q_v and *d(qp,* v_x*)* is smaller than the distance stored, update the distance to *d(qp,* v_x*)*, else do nothing;
(7) Repeat steps 5 and 6 until finding m data points.

Using heuristic function, we make the search more intelligent to the virtual data point so that it can reduce the traversed vertex. Figure 3 is an Island algorithm’s approximate path extension diagram and Figure 4 is the improved Island algorithm’s approximate path extension diagram.

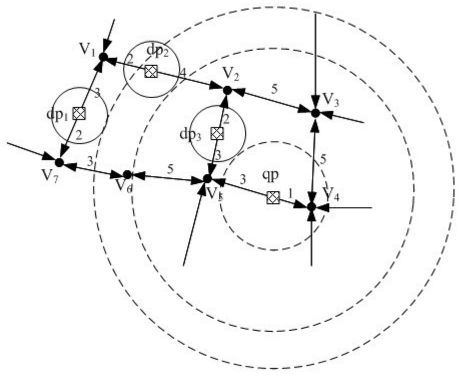


Figure 3. Island Algorithm’s Approximate Path

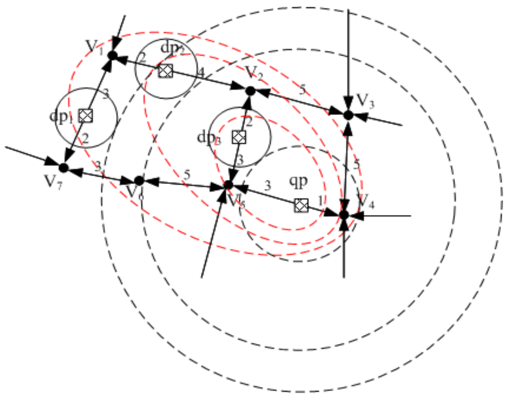


Figure 4. Improved Island Algorithm’s Approximate Path

The figures show that the improved network traverses fewer vertexes than original one. When k=1, for example, suppose the shortest path length is 2a, so the maximum search radius of the Island is 2a and the major axis of the ellipse of improved Island is 2a. Search area as follows:

The area of circle: $S = 4\pi a^2$

The area of ellipse: $S = \pi a^2\sqrt{1-e^2}, e = c/a, c < a$

When k=1, Island algorithm’s maximum search area is 4 times more than improved Island algorithm’s. Of course, the improved algorithm is faster which can be verified from the following experimental data.

4.2. Data Structure

Island algorithm does not return query path and can’t be directly applied to path planning scenarios, so it is necessary to optimize the data structure. Designed data structure is based on Java can be seen in Figure 5.

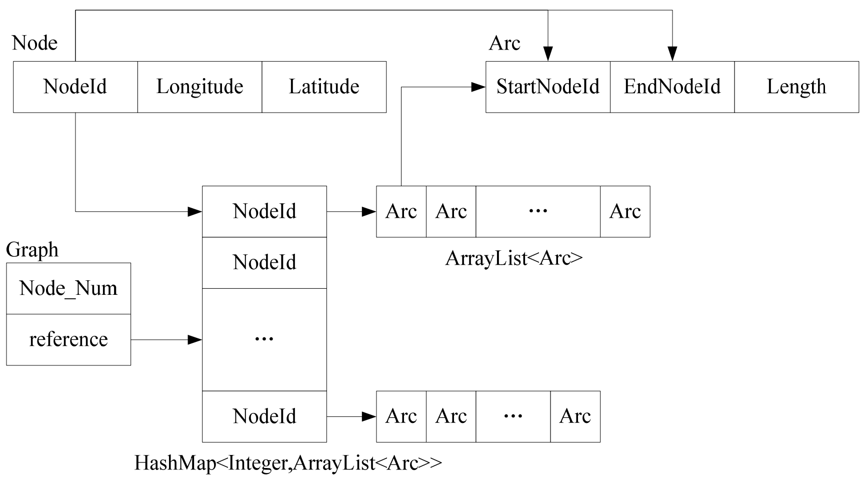


Figure 5. Graph Structure

We use the adjacency list as graph storage structure, meanwhile, using HashMap data structure to speed up the search for set of adjacent edges. The details of graph storage structure can be seen in Figure 6.

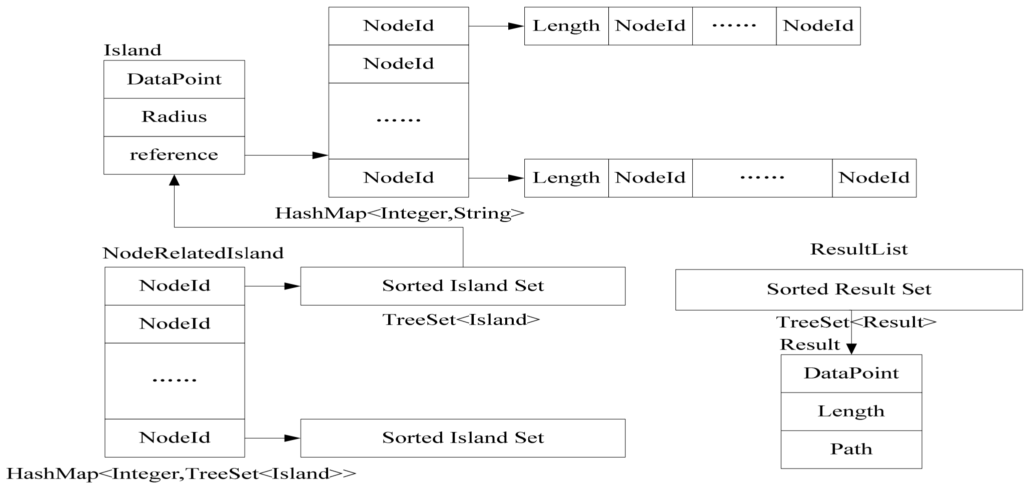


Figure 6. The Main Storage Structure

Island class stores pre-computed results which stored in file in default and loaded into memory when needed. The storage form of Island class is *(v_x, $d(v_x,dp)$, path)*, v_x is the Island's vertex, *$d(v_x,dp)$* is the shortest distance from *dp* to v_x and *path* stores detail shortest path. NodeRelatedIsland class stores vertex's Island reference, if the vertex is covered by Island, there must exists reference in NodeRelatedIsland class. ResultList class contains the nearest data points and the shortest path.

According to the above algorithm, first we should search the NodeRelatedIsland class to judge whether *$qp.e.v_e$* has Island references which cover it and put Island references into ResultList class. If ResultList.size()>=k, the search terminates, otherwise start the network expansion. Visit the graph and find *$qp.e.v_e$*'s adjacent edges. We replace the Arc.length with *$f(v_x)$* and put the Arc into the priority queue. Get priority queue's first element, remove it and search the NodeRelatedIsland to judge whether Arc.EndNodeId has Island references. And then regard Arc.EndNodeId as the next search vertex. Repeat the process until we find the k nearest data points. The id and shortest path of nearest data points can be found in ResultList class.

The structure uses a lot of HashMap and TreeSet. HashMap as mentioned before can improve the search efficiency. TreeSet based on Red-Black Tree [10] which is the self-balancing binary search tree. Its operation's complexity is *$O(\log_2 n)$*. We can use comparator to make the red-black tree maintains an orderly state. Priority queue needs frequent insert and delete operations, so let TreeSet as priority queue is a wise choice. As can be seen, the optimized data structure is able to return the shortest path and the complexity of improved Island algorithm network expansion is *$O(n\log_2 n)$*.

4.3. Simulation and Analysis

The experiment uses Shanghai road network. After modeling, the graph contains 2014740 vertexes and 2037521 adjacent edges. Hardware configuration: Intel (R) Core (TM) 2 Duo CPU, 2.0GB memory. Java is an algorithm implementation language and Java virtual machine maximum heap memory is 512MB. Query point and data points' distribution as shown in the Figure 7.

The circle indicates pre-computed area and arrow indicates the direction of movement of query point. Add direction constraints that means exclude *{dp1, dp8, dp9, dp10, dp11}* as the result. We analyze performance from the number of traversal vertexes and the execution time. The results are in Figure 8 and Figure 9.

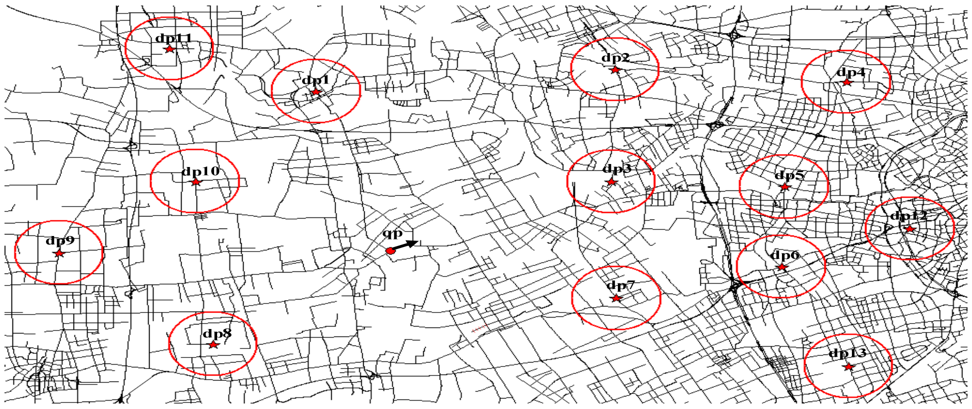
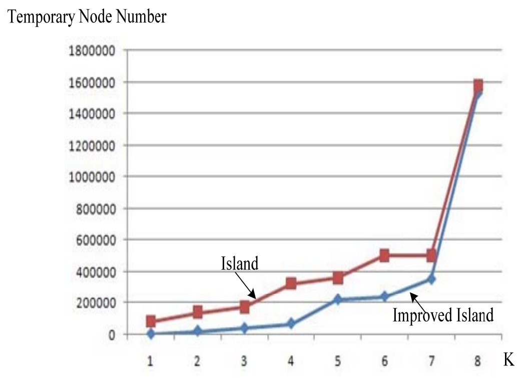


Figure 7. Query Point and Data Points' Distribution in Road Network



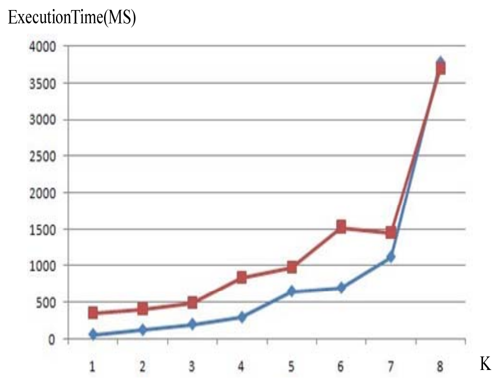


Figure 8. The Number of Traversal Vertexes

Figure 9. Execution Time

When k=2, for example, the experimental parameters are shown in Table 1, and the results are in Figure 10.

Table 1. Algorithm Performance Comparison when k=2

Name	Execution Time (MS)	Temporary Node Number	Result
Improved Island	125	22516	(dp3, 9744.26) (dp7, 9787.76)
Island	407	13849	(dp3, 8546.77) (dp7, 9735.81)

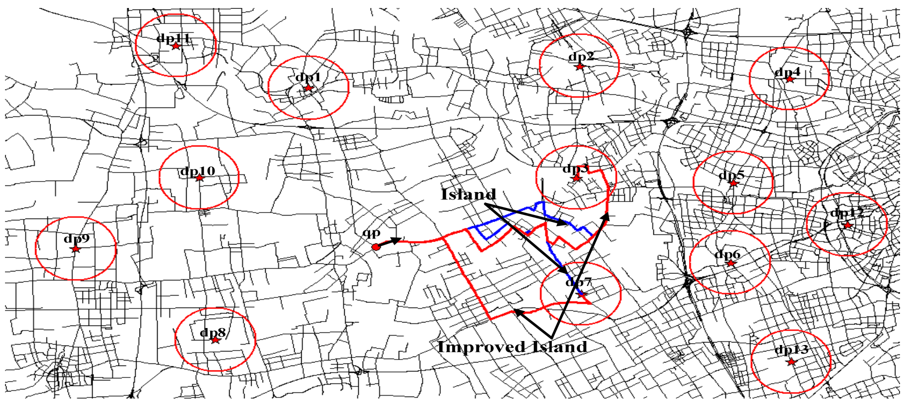


Figure 10. The Result when k=2

In summary, improved Island algorithm traverses fewer vertexes than original one and more effective. Although the improved algorithm cannot give optimal solution, it has a high query performance and more suitable for high real-time demanding scenarios.

5. Application Example

The improved Island algorithm can be applied in the electric vehicle charging guidance system which has high real-time requirement.

Now compared to the gas station, electric vehicle power distribution infrastructure resources are scarce and unevenly distributed. At the same time, charging station planning research is still in its infancy at home and abroad [11]. It is likely that electric vehicle in use process needs to be recharged but couldn't find the charging station. On the one hand, a perfect battery monitoring technology [12] is important, on the other hand, the electric vehicle route guidance is also very necessary. The guidance system must provide real-time route tips, or when the vehicle away from the location, the given path loss its meaning.

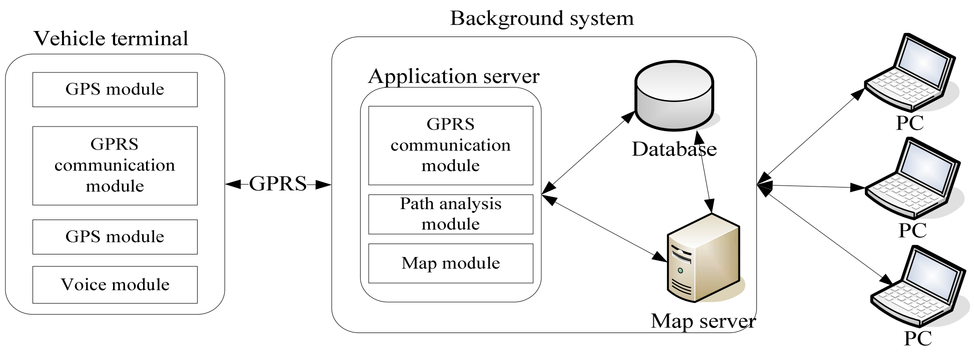


Figure 11. The Electric Vehicle Charging Guide System

When electric vehicle battery is lower than the warning value, the current GPS and power information will be sent to the background system. The background system calculates the path and sends it to the vehicle terminal. And the terminal guides the vehicle by voice prompts to the nearest charging station. The background system will continue to monitor the vehicle, when it deviates from the given driving route, system re-calculates the path until the vehicle reaches the charging station.

Electric vehicle charging guidance system is k=1 nearest neighbor query problem. The improved Island algorithm's query performance is shown in the Figure 12. Here we assume Island radius is 1500 meters, using the hardware environment as described in 4.3.

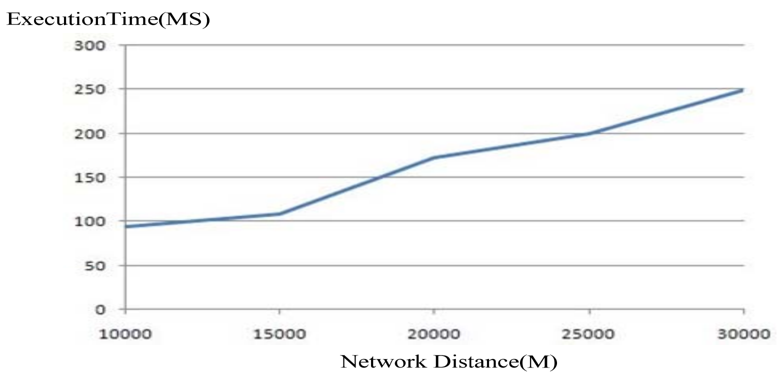


Figure 12. The Performance when k=1

As illustrated above, find the nearest charging station in 30KM scope uses less than 300ms. If electric vehicle's speed is 60KM/h, the traveling distance of the vehicle in query process will be less than 5 meters. So the offset is small and meet the real-time requirement.

6. Conclusion

This paper proposes an improved Island algorithm. We take the best data point estimate and heuristic search method, effectively reducing the traversal vertexes. Besides, we optimize the data structure. The experimental results show that the improved Island algorithm makes the query performance more effectively and can return the nearest neighbor paths, is suitable for the high real-time demanding route guidance applications. This paper doesn't discuss the pre-computed Island radius and the algorithm based on static road network using length as the road priority. The road priority in real life should also take the degree of congestion and road grade into account. Therefore, the research emphasis in the future will be the improvement of the Island algorithm under dynamic network, as well as the Island pre-computed radius determining strategy.

References

[1] Hu L, Shahabi C, Bakiras S. Spatial Query Integrity with Voronoi Neighbors. *Knowledge and Data Engineering IEEE Transactions on*. 2013; 25(4): 863-876.

[2] Guobin Li, Jine Tang. *A New K-neighbor Search Algorithm Based on Variable Incremental Dynamic Grid Division*. Computational Intelligence and Design (ISCID). Hangzhou. 2010; 2: 167-170.

[3] Ya Sun. Design and Implementation of Nearest Neighbor Query in Spatial Network Database. *Computer Science*. 2008; 35(3):73-75.

[4] Jingfeng Guo, Hanfeng Liu, Qian Ma. K Nearest Neighbor Queries of Moving Objects Based on Networks. *Computer Engineering*. 2008; 34(3): 100-104.

[5] Papadias D, Zhang J, Mamoulis N. *Query processing in spatial network database*. Proceedings of the 29th International Conference on Very Large Data Bases (VLDB). 2003; 29: 802-813.

[6] Kolahdouzan M, Shahabi C. *Voronoi-based K nearest neighbor search for spatial network databases*. Proceedings of the 30th International Conference on Very Large Data Bases (VLDB). 2004; 30: 840-851.

[7] Huang X, Simonas S. *The Islands Approach to Nearest Neighbor Querying in Spatial Networks*. Proceeding of the 9th International Symposium on Spatial and Temporal Data Bases (SSTD). 2005: 73-90.

[8] Wang H, Zimmermann R. *Location-based Query Processing on Moving Objects in Road Networks*. Proceedings of the 30th International Conference on Very Large Data Bases (VLDB). 2007.

[9] Hyung Ju Cho, ChinWan Chung. *An efficient and scalable approach to CNN Queries in a Road Network*. Proceedings of the 31th International Conference on Very Large Data Bases (VLDB). 2005: 865-876.

[10] Jianwei Li, Yubin Xu, Hong Guo. *Memory Organization in a Real-time Database Based on Red-black Tree Structure*. Intelligent Control and Automation. 2004; 5: 3971-3974.

[11] Hanwu Luo, Fang Li. A Method for Electric Vehicle Ownership Forecast Considering Different Economic Factors. *TELKOMNIKA Indonesia Journal of Electrical Engineering*. 2013; 11(4): 2239-2246.

[12] Lei Lin, YuanKai Liu, Wang Ping, Fang Hong. The Electric Vehicle Lithium Battery Monitoring System. *TELKOMNIKA Indonesia Journal of Electrical Engineering*. 2013; 11(4): 2247-2252.