Towards Efficient and Flexible KNN Query Processing in Real-Life Road Networks

Yang Lu† Bin Cui† Jiakui Zhao†
† Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, CHINA
School of Electronics Engineering and Computer Science, Peking University, CHINA
{yanglu,bin.cui,jkzhao}@pku.edu.cn

Hua Lu‡
‡Aalborg University
luhua@cs.aau.dk

Jialie Shen§
§Singapore Management University
jlshen@smu.edu.sg

Abstract

Along with the developments of mobile services, effectively modeling road networks and efficiently indexing and querying network constrained objects has become a challenging problem. In this paper, we first introduce a road network model which captures real-life road networks better than previous models. Then, based on the proposed model, we propose a novel index named the RNG (Road Network Grid) index for accelerating KNN queries and continuous KNN queries over road network constrained data points. In contrast to conventional methods, speed limitations and blocking information of roads are included into the RNG index, which enables the index to support both distance-based and time-based KNN queries and continuous KNN queries. Our work extends previous ones by taking into account more practical scenarios, such as complexities in real-life road networks and time-based KNN queries. Extensive experimental study shows that our methods are efficient in terms of both CPU time and disk I/Os.

1. Introduction

The continuous developments of mobile computing, positioning technologies, and wireless communication technologies have made it possible and feasible to track and record the positions of continuously moving objects. As a result, mobile services are becoming increasingly practical and important, within the big picture of spatiotemporal data management. The main challenge of mobile services is how to efficiently deliver required data objects to corresponding mobile users who issue location dependent queries. In real-life applications, users usually move inside a spatial network, e.g., a road network. They often issue $K$ Nearest-Neighbor (KNN) or Continuous KNN (CKNN) queries to obtain interesting data objects reachable through the road network.

Previous works study KNN queries have mainly been done in the Euclidean space. Whereas the case in the road network context is more complicated. As users can only move inside pre-defined road networks, the nearest neighbors in the Euclidean space usually are not the nearest neighbors in road networks. The research works on road networks are mainly focused on two topics, i.e., modeling road networks [4, 5, 6] and indexing and query processing [7, 10, 13]. Nevertheless, current CKNN queries over road networks are all distance-based, i.e. the distance between the user and an object in the road network is used as the metric. Some other important metrics like the estimated time for a user to reach an object are not fully considered. In addition, important real-life road network complexities like road directions, U-turns, and speed constraints are not considered, though they have significant influences on the query results.

Motivated by above observations, in this paper, we first design a road network model which captures real-life road networks better than previous models. Then, based on the proposed model, we propose a novel index for accelerating KNN and CKNN queries over road networks. Our contributions are summarized as follows:

1. We design a road network model which takes into account most important characteristics of real-life road...
networks. Such characteristics include road types, directions, speed limitations, and blocking information on roads. As a result, our work based on this model applies to most real-life application scenarios.

2. We propose an index named the Road Network Grid (RNG) index for accelerating KNN and CKNN queries over road networks. Since the density of roads is typically highly skewed, we utilize the quad-tree instead of the uniform grid structure to partition the road networks.

3. We develop efficient methods for processing both time-based and distance-based KNN and CKNN queries using the RNG index built over our network model.

4. We conduct extensive experimental study to demonstrate that our methods are efficient on both CPU time and disk I/Os.

The rest of this paper is organized as follows. Section 2 summarizes related works. Section 3 introduces our road network model. Section 4 presents the RNG index and the methods for processing KNN and CKNN queries, followed by the performance evaluation in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

Due to the importance in practice, the KNN and CKNN queries in road networks have gained considerable research efforts in recent years [7, 8, 9, 11, 14]. In such contexts, the network distance instead of the Euclidean distance becomes the metric, which requires special process.

The Incremental Network Expansion (INE) method [14] computes KNN on-the-fly by adapting the Dijkstra algorithm [3]. However, if the objects are sparse, the method has to explore large part of the network for retrieving KNNs. Kolahdouzan et al. [13] proposed the VN\textsuperscript{3} method, which does not incrementally explore the road network but checks pre-computed Voronoi diagram to guide KNN search. Experiments show that the VN\textsuperscript{3} method outperforms the INE method by up to one order of magnitude in response time. But, the VN\textsuperscript{3} method suffers from high pre-computation overhead for each Voronoi region when the dataset gets denser. Hu et al. [8] proposed an index for simplified road networks to accelerate KNN queries. Huang et al. proposed the islands method [10] and the S-GRID method [9] which pre-compute and store network distance information to facilitate query processing.

Continuous KNN (CKNN) queries in networks were first studied by Kolahdouzan et al. [12] with two processing methods. The IE method divides pre-defined path into some segments, and candidate KNNs is generated by merging the KNNs of the vertices of segments. The UBA method improves the IE method by determining a bound during which no split points exist. However, the performance of both methods is greatly influenced by the object density. To solve this problem, Cho and Chung [2] proposed a method called UNICONS for processing KNN and CKNN queries, which divides the path into subpaths, determines valid intervals for each subpath, and merges intervals of adjacent subpaths.

3. Modeling Real-Life Road Networks

To the best of our knowledge, existing KNN and CKNN queries over road networks are all based on a simple road network model which is actually an undirected graph, where junctions and roads in real-life road networks are mapped into vertexes and undirected edges respectively. However, that model does not accurately characterizes real-life road networks. Real-life roads are often double-track roads, U-turns are allowed only if the line between two tracks is dotted line, and an object is only located at one side of the road.

On the other hand, the metric used by existing KNN and CKNN queries over road networks is the distance only. In some cases, however, time may be more important than distance. The distance-based nearest neighbors usually are not the time-based nearest neighbors. In order to support time-based KNN and CKNN queries, the speed limitations and the blocking information of roads should be included into the road network model.

Motivated by these observations, we proceed to propose a road network model which captures real-life road networks better than previous approaches. First of all, we divide all roads in a real-life network into four types. They are defined in Definitions 1 to 4, where the blocking factor is included to reflect the traffic conditions like congestions.

Based on the four road types, we will show how to represent a real-life road network in our road network model.

Definition 1 First-Grade Road: A single track road is a first-grade road, and can be characterized as

$$\text{Road}_1 = \{\text{rid}, v_s, v_e, \text{len}, \text{dp}, v_{sp}, \text{bf}\}$$

where \text{rid} is the identifier of the road, \(v_s\) and \(v_e\) are the start point and the end point of the road respectively, \text{len} is the length of the road, \text{dp} is the set of objects which are located at both sides of the road, \(v_{sp}\) is the maximal speed limitation of the road, and \text{bf} is the blocking factor of the road.

Definition 2 Second-Grade Road: If a road is a double track road and U-turn is not allowed, it is a second-grade road, and can be characterized as

$$\text{Road}_2 = \{\text{rid}, v_s, v_e, \text{len}, \text{dp}, \overrightarrow{v_{sp}}, \overrightarrow{\text{bf}}, \overrightarrow{\text{dp}}, \overrightarrow{v_{sp}}, \overrightarrow{\text{bf}}\}$$
A road is a double track road, and U-turn is allowed only at some parts of the road, it is a third-grade road.

**Definition 3 Third-Grade Road:** If a road is a double track road, and U-turn is allowed only at some parts of the road, it is a third-grade road.

**Definition 4 Fourth-Grade Road:** If a road is a double track road and U-turn is allowed throughout the road, it is a fourth-grade road, and can be characterized as

\[
\text{Road}_4 = \{\text{rid}, v_s, v_e, \text{len}, dp, v_{sp}, \text{bf}, \text{bf}^f\}
\]

where the parameters are the same as those defined in Definition 1 and Definition 2.

Actually, a second-grade road can be regarded as two first-grade roads, and a third-grade road can be regarded as several second-grade and fourth-grade roads. Figure 1(a) shows an example of a real-life road network. Here, \(v_5v_3\) is a first-grade road; \(v_1v_5\) is a second-grade road; \(v_4v_5\) is a third-grade road; \(v_4v_5\) is a fourth-grade road. Data objects \(dp_1, dp_2, \ldots, dp_7\) are located at one side of a road, and \(v_5\) is the junction of four roads.

Given a real-life road network, we model it as a mixed graph \(G = (V, E)\), where \(V\) and \(E\) are the set of vertices and the set of edges respectively. Each vertex \(v = (\text{vid}, \text{pos})\) in \(V\) corresponds to a junction in the real road network, where \(\text{vid}\) is the identifier of the vertex and \(\text{pos}\) is the position of the vertex in the 2-dimensional space. Each first-grade road corresponds to an directed edge, each second-grade road corresponds to two directed edges, each fourth-grade road corresponds to an undirected edge, and each third-grade road is partitioned into several second-grade and fourth-grade roads. Each directed edge \(e\) is characterized as

\[
e = \{\text{eid}, v_s, v_e, \text{len}, dp, v_{sp}, \text{bf}\}
\]

and each undirected edge \(e\) can be characterized as

\[
e = \{\text{eid}, v_s, v_e, \text{len}, dp, v_{sp}, \text{bf}^f, \text{bf}^f\}
\]

where \(\text{eid}\) is the identifier of an edge, and other parameters are the same as those in Definition 1 and Definition 2.

Figure 1(b) shows the mixed graph model corresponding to the real-life road network shown in Figure 1(a). Each junction is mapped into a vertex. The first-grade road \(v_5v_3\) is mapped to a directed edge \(e_4\). The second-grade road \(v_1v_5\) is mapped into two directed edges \(e_1\) and \(e_2\). The third-grade road \(v_5v_4\) is partitioned into three consecutive segments: two second-grade roads and one fourth-grade road. The two second-grade roads are mapped to four directed edges \(e_5\), \(e_6\), \(e_8\) and \(e_9\), and the fourth-grade road is mapped to an undirected edge \(e_7\). The fourth-grade road \(v_5v_2\) left is mapped to an undirected edge \(e_3\).

4. Indexing and Query Processing

In this section, we first introduce the RNG index, and then present how to process KNN and CKNN queries using the RNG index.

4.1. The RNG Index

The recent S-GRID method [9] partitions a road network into grids of the same cell size. However, different areas in the road network usually have different vertex densities, which is an important factor that influences the performance of query processing. When a query object issues a query, the cell which contains the object needs to be determined. In order to locate the query object in the cell, vertices and edges in the cell will be accessed. If a cell contains many vertices, considerable CPU time and disk I/Os will be spent on locating the query object. To overcome this problem, in this paper, we will propose a partition strategy intended to make each cell contain similar number of vertices, and thus speed up the query object locating.

Along with our partition, we generate a quad-tree to index all cells produced by the partition. In particular, our partition strategy works as follows. First, we confine the whole area of interest using a rectangle, i.e. a large cell, upon the road network model we obtain in the way presented in Section 3. This cell corresponds to the root node of the quad-tree. If the number of the vertices inside a cell is larger than a threshold \(\gamma\), the cell will be further divided into four subcells with the same size. The process recurs until
the number of vertices in root of edge len $G$ corresponds to an edge in the road network, we use a tag $f$ to distinguish directed edges and undirected edges. If $f = 0$, $vv'$ is a directed edge. If this directed edge corresponds to a first-grade road, $dp$ stores all the objects located at both sides of the road. If this directed edge corresponds to a track of a second-grade road, $dp$ stores all the objects which locate at the side along the track. For an undirected edge, $dp$ stores all the objects which locate at both sides of the corresponding fourth-grade road, and the same $dp$ can be shared by undirected edges $vv'$ and $v'v$. If $f = 1$, $vv'$ is an undirected edge and the objects in $dp$ are arrayed along $vv'$. If $f = 2$, $vv'$ is an undirected edge and the objects in $dp$ are arrayed along $v'v$.

The algorithm for building the RNG index is shown in Algorithm 1. It requires the mixed graph model $G$ and the threshold value $\gamma$ as inputs, and returns the root of the quad-tree generated during the partition. The generated quad-tree helps to quickly locate the query object, and thus accelerating KNN and CKNN query processing.

4.2. Processing KNN Queries

In this section, we address how to process KNN queries in which data objects of interest are static and distributed along roads and the query object is given as a fixed location inside the road network. The edges in our road network model may be either directed or undirected, which differs from previous works that only consider undirected edges. Thus the nearest neigh-

![Figure 2. An example of RNG index.](image)

Figure 2. An example of RNG index.

![Figure 3. Information of adjacent edges.](image)

Figure 3. Information of adjacent edges.

none of the cells contains more than $\gamma$ vertices. Meanwhile, the quad-tree grows higher.

Figure 2(a) shows an example of our partition, and Figure 2(b) the corresponding quad-tree. For the grid partition shown in Figure 2(a), the threshold $\gamma$ is set to 5. Because cell2 contains 6 vertices, it is further partitioned into four subcells.

In the quad-tree, each inner node stores the corresponding cell positions as well as four subcell pointers. While each leaf node stores the corresponding cell positions as well as the information of all adjacent edges as shown in Figure 3.

In Figure 3, a vertex list $vl$ stores the positions of all vertices inside the cell, and each vertex points to an adjacent vertex list $avl$ which records positions of adjacent vertices. Each entry of the $avl$ corresponds to an edge in the road network model, and the length, speed limitation, and blocking factor of an edge are all stored in the entry. In addition, each entry of $avl$ has a pointer to an object list $dp$ which stores positions of all the objects located at the corresponding edge.

Since a vertex $v$ in $vl$ and a vertex $v'$ in $avl$ determine an edge $vv'$ in road network model, we use a tag $f$ to distinguish directed edges and undirected edges. If $f = 0$, $vv'$ is a directed edge. If this directed edge corresponds to a first-grade road, $dp$ stores all the objects located at both sides of the road. If this directed edge corresponds to a track of a second-grade road, $dp$ stores all the objects which locate at the side along the track. For an undirected edge, $dp$ stores all the objects which locate at both sides of the corresponding fourth-grade road, and the same $dp$ can be shared by undirected edges $vv'$ and $v'v$. If $f = 1$, $vv'$ is an undirected edge and the objects in $dp$ are arrayed along $vv'$. If $f = 2$, $vv'$ is an undirected edge and the objects in $dp$ are arrayed along $v'v$.

The algorithm for building the RNG index is shown in Algorithm 1. It requires the mixed graph model $G$ and the threshold value $\gamma$ as inputs, and returns the root of the quad-tree generated during the partition. The generated quad-tree helps to quickly locate the query object, and thus accelerating KNN and CKNN query processing.

4.2. Processing KNN Queries

In this section, we address how to process KNN queries in which data objects of interest are static and distributed along roads and the query object is given as a fixed location inside the road network. The edges in our road network model may be either directed or undirected, which differs from previous works that only consider undirected edges. Thus the nearest neigh-

Algorithm 1: Building RNG Index($G$, root, $\gamma$)

Input: $G$: the road network model
$\gamma$: the maximal number of vertices in a cell
Output: root: the root of the generated quad-tree

begin
if the number of vertices in $G \leq \gamma$ then
	foreach vertex $v$ in $G$ do
		foreach adjacent vertex $v'$ of $v$ do
			hook each object $o$ on edge $vv'$ do
			insert $o$ into $dp$ in ascending order of offset;
		end
end
else
	if $dp$ of $v'v$ has not been built then
		$f := 1$;
		foreach object $o$ on edge $vv'$ do
			insert $o$ into $dp$ in ascending order of offset;
	end
else
	$f := 2$;
	$dp :=$ the $dp$ of $v'v$;
end

root.$vl$.addEntry($v$);

$avl$.addEntry($v'$, len, $v_a$, $bf$, $f$, $dp$);
end

end
bors are probed along road directions. This is quite different from previous methods which search nearest neighbors along arbitrary road directions. In addition, we consider both distance-based and time-based KNN queries. We extend the INE method [14] for processing KNN queries, by using RNG index to speed up the query object locating and the query processing.

In previous works, distance is the only metric for KNN queries. As a matter of fact, time can be more important than distance in some cases. For example, a user may care more about how long it will take him to reach a restaurant, instead of the distance to the nearest restaurant. Equation 1 characterizes the estimated time \( t \) for a moving object to move length \( len \) along a road, where \( v \) is the speed of the moving object whose value must be smaller than the speed limitation \( v_{sp} \), and \( bf \) is the blocking factor of the road whose value can be acquired form a road sensor. The value of \( bf \) is between 0 and 1, which increases if the traffic jam becomes more severe. The value of \( bf \) is not updated frequently, especially in the congestion case, and thereby it can be regarded as static during the processing of a KNN query.

\[
t = \frac{len}{v \ast (1 - bf)}
\]

Algorithm 2 shows our method for processing KNN queries using the RNG index, which supports both distance-based and time-based KNN queries. To make the presentation simple, we assume that the algorithm is distance based in the rest of this section. To change the algorithm to a time based one, we only need to use the estimated time defined by Equation 1 as the metric instead. When a query is issued, the number of the required nearest neighbors \( k \) along with the query object \( q \) and the two vertices of the edge where the query is issued, \( v_s \) and \( v_e \), are submitted. If the edge is a directed edge, \( v_s \) and \( v_e \) are the start vertex and the end vertex respectively. Otherwise, \( v_s \) and \( v_e \) are the two vertices. An array \( rq \) stores the objects being found currently, where objects are kept in an ascending order of the distance to the query object. Therefore, the first \( k \) objects are the KNNs. A priority queue \( cq \) stores the vertices which will be used for further exploring, and the \textit{dequeue}() operation returns the vertex which is nearest to the query object. Initially, \( rq \) and \( cq \) are set to empty. The \textit{findavr}() function is used to find the avl entry in the RNG index which corresponds to the edge where the query is issued. Using the quad-tree, the \textit{avl} entry can be found efficiently. If the edge is an undirected edge (lines 5-9 in Algorithm 2), the \textit{get_all_objects} function is used to get all the objects along the edge; the objects are stored in \( oq \) and kept in an ascending order of the distance to the query object. Then, the two vertices of the edge along with the distances to the query object are enqueued into \( cq \) for further exploration. If the edge is a directed edge, the \textit{getsome_objects} function is used to get the objects from the position where the query is issued to the end of the edge. These objects are stored in \( oq \) and kept in an ascending order of the distance to the query object. Then, if the number of objects in \( oq \) is not less than \( k \), we select the first \( k \) objects from \( oq \) and return...
4.3. Processing CKNN Queries

Previous CKNN queries [2, 12] maintain the KNNs during the moving of the query object along a pre-defined path. The pre-defined path may be divided into some intervals and the KNNs are static when the query object moves inside the interval, and hence the query result is a set of intervals and the corresponding KNNs. However, the assumption that the query object moves along pre-defined path is not practical, because the query object may dynamically choose the path according to traffic conditions, accidents, etc. Motivated by this, in this section, we proceed to address how to process CKNN queries in the dynamic environment where the query object moving in the network can freely choose the path in an on-line manner.

Algorithm 4 shows our method for processing CKNN queries. The input, output, variables, and functions are the same as those in Algorithm 2. When a CKNN query is submitted, the $avl$ entry corresponding to the edge where the query is issued is found via the RNG index. The query will continuously run until it is explicitly canceled. Each time the query object moves to a new edge, the corresponding $avl$ entry can be found efficiently using the RNG index. If the query object moves on a directed edge, i.e., $avl_entry.f = 0$, the objects along the edge from the query object to the edge end are obtained as candidate KNNs, and current KNNs can be selected from candidate KNNs. If the query object passes by an object, the object will be deleted from the candidate KNNs and current KNNs, and a new object, i.e., the $k$th nearest neighbor, will be selected from candidate KNNs as part of current KNNs.

If the query object moves to an undirected edge, the objects on the edge to both directions of the query object are obtained as candidate KNNs. Then, the edge is divided into some intervals. While the query object moves inside an interval the KNNs result does not change. Hence, the current KNNs need to be updated only when the query object moves from one interval to another. The method proposed in [2] can be extended to find the connection points between intervals. Note that the CKNN queries in [2] only allow pre-defined paths of moving query objects.

5. Performance Evaluation

In this section, we evaluate the performance of our methods by extensive experimental study. A real-life dataset, the road network of San Francisco [1], is used in our experiments. The dataset contains 174,956 vertices and 223,001 edges. Since second-grade roads are very common in practice, 50% roads in the dataset are set as second-grade roads, 25% roads are set as first-grade roads, and the remaining 25% roads are set as third/fourth-grade roads. The real-life road networks are converted to our road network model. The queried objects are randomly generated with specified density. The speed limitation and the blocking factor of each road are also randomly generated, as well as the query location and the query speed.

We compare the proposed method with the S-GRID [9]
method which shows better performance than existing methods such as Island [10] and INE [14]. For each set of experiments, we run 1,000 queries and measure the average performance. The page size is set to 4KB and a LRU buffer is used. The algorithms are implemented in the C++ language and run on a 1.8GHz AMD CPU with 2GB memory running Windows Advanced Server 2003.

5.1. Experiments for KNN Queries

In order to compare our method with the S-GRID method, for each query, we first use our method to find the $k$ nearest neighbors under our road network model. Then we apply the $k$-th NN distance in the KNN query processing of the S-GRID method to find the $k^c$ nearest neighbors under the road network model which only contains undirected edges. Generally, the $k^c$ nearest neighbors cover our $k$ nearest neighbors, and $k^c$ is larger than $k$. We compare the two methods using the average CPU time and disk I/Os for processing 1,000 queries. Since the S-GRID approach does not support time-based KNN queries, we will not compare our method with the S-GRID method for time-based KNN queries. In all experiments, the S-GRID approach partitions the road networks into $20 \times 20$ cells.

In the first set of experiments, we test the performance under different threshold values (i.e. the maximal number of vertices in a cell). The density of objects (i.e. the ratio between the number of objects and the number of the edges) is set to 0.05 and the number of the nearest neighbors is set to 3; the threshold value increases from 50 to 2000. Figure 4(a) and Figure 4(b) show the experimental results. The CPU time and the disk I/Os are determined by the number of non-leaf nodes and vertices visited in the leaf nodes. If the threshold value is very small, many non-leaf nodes have to be examined. However, if we increase the threshold value, more vertices have to be examined in a certain cell. We can see that when the threshold value is relatively small, e.g. 100-500, both CPU time and disk I/Os are near optimal. For clarity of presentation, we will fix the threshold value to 100 in the following sets of experiments.

In the second set of experiments, we test the performance under different numbers of nearest neighbors, i.e. varying $k$. The density of objects is set to 0.3, and the number of nearest neighbors increases from 5 to 40. Figure 4(c) and Figure 4(d) show the experimental results. The CPU time and disk I/Os of both methods increase with the increase of $k$ because a larger number of edges are visited. Our method outperforms the S-GRID method on both CPU time and disk I/Os, because it visits much fewer non-leaf nodes and vertices in leaf nodes. Note that, the $k$ nearest neighbors are evaluated in our network model with directed paths.

Therefore, S-GRID may need to examine more neighboring objects to find the exact KNNs.

Next, we study the performance under different object densities. The number of the nearest neighbors is set to 5, and the object density increases from 0.1 to 0.5. Figure 4(e) and Figure 4(f) show the experimental results. The CPU time and disk I/Os of both methods decrease with the increase of the object density due to smaller number of edges visited. Our method outperforms the S-GRID method on both CPU time and disk I/Os, because it visits smaller number of non-leaf nodes and vertices in leaf nodes.

Finally, we investigate the relationship between distance-based KNNs and time-based KNNs. In our approach, each road may have different speed limits or blocking factors, thus short distance does not mean fast time. Therefore, we want to compare the effect on KNNs in terms of two metrics, i.e. distance and time. We define the hit rate of the time-based KNNs as the ratio between the number of the time-based KNN queries returning the same KNNs as the distance-based KNN queries, and the number of queries issued. First, we set the object density to 0.6 and increase the number of nearest numbers from 2 to 10. Figure 4(g) shows the experimental results. The hit rate decreases with the increase of the number of nearest neighbors as many edges need to be visited and the speed limitations and blocking factors have great influence on the nearest neighbors. Second, we set the number of the nearest neighbors to 6 and increase the object density from 0.05 to 0.25. Figure 4(h) shows the experimental results. The hit rate increases with the increase of the object density because fewer edges need to be visited and the speed limitations and blocking factors have small influence on the nearest neighbors.

5.2. Experiments for CKNN Queries

We focus on CKNN queries in this subsection. Given a query object, we first find the $k$ nearest neighbors in the road network, then update the KNNs according to the movement of the query object. We compare our method with the S-GRID method in the same way as in Section 5.1. The number of roads in the path chosen by the query object in the online manner is set as a varying parameter. In addition, the threshold value is set to 100.

We first test the performance under different numbers of edges in the path. The number of the nearest neighbors is set to 3 and the object density is set to 0.5. The number of the edges in the path increases from 2 to 10. From Figure 5, we see that our method outperforms the S-GRID method on both CPU time and disk I/Os, because it visits smaller number of non-leaf nodes and vertices. We also test the performance under different object densities and numbers of nearest neighbors, and the results are similar to that of KNN queries. We omit the details due to page limits.
6. Conclusions

In this paper, we first designed a road network model which characterizes real-life road networks better than previous models. Based on that model, we proposed the novel RNG index to accelerate the processing of KNN and CKNN queries. The RNG index partitions the road networks into cells with different sizes according to the density of vertices, and the speed limitations and the blocking factors of roads are included into the index. As a result, our method supports both distance-based and time-based KNN and CKNN queries. We also developed robust algorithms for processing KNN and CKNN queries using the RNG index. Our CKNN query algorithm allows the query object to choose the path in an online-manner. Extensive experimental study demonstrates that our methods outperforms the state-of-the-art method, i.e. the S-GRID method, in terms of both CPU time and disk I/Os.

References