International Journal of Geo-Information

MDPI

Article

A Line Graph-Based Continuous Range Query Method for Moving Objects in Networks

Hengcai Zhang [1,2], Feng Lu [1,2,3,*] and Jie Chen [1]

[1] State Key Lab of Resources and Environmental Information System, Institute of Geographic Sciences and Natural Resources Research, Chinese Academy of Sciences, Beijing 100101, China; zhanghc@lreis.ac.cn (H.Z.); chenj@lreis.ac.cn (J.C.)

[2] Fujian Collaborative Innovation Center for Big Data Applications in Governments, Fuzhou 350003, China

[3] Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing 210023, China

* Correspondence: luf@lreis.ac.cn; Tel.: +86-10-6488-8966

Academic Editors: Georg Gartner, Haosheng Huang and Wolfgang Kainz

Received: 31 May 2016; Accepted: 13 December 2016; Published: 19 December 2016

Abstract: The rapid growth of location-based services has motivated the development of continuous range queries in networks. Existing query algorithms usually adopt an expansion tree to reuse the previous query results to get better efficiency. However, the high maintenance costs of the traditional expansion tree lead to a sharp efficiency decrease. In this paper, we propose a line graph-based continuous range (LGCR) query algorithm for moving objects in networks, which is characterized by a novel graph-based expansion tree (GET) structure used to monitor queries in an incremental manner. In particular, GET is developed based on the line graph model of networks and simultaneously supports offline pre-computation to better adapt our proposed algorithm to different sizes of networks. To improve performance, we create a series of related data structures, such as bridgeable edges and distance edges. Correspondingly, we develop several algorithms, including initialization, insertion of objects, filter and refinement and location update, to incrementally re-evaluate continuous range queries. Finally, we implement the GET and related algorithms in the native graph database Neo4J. We conduct experiments using real-world networks and simulated moving objects and compare the proposed LGCR with the existing classical algorithm to verify its effectiveness and demonstrate its greater efficiency.

Keywords: moving objects; continuous range queries; network; expansion tree

1. Introduction

With the advancement of wireless networks and the development of positioning technologies, such as GPS, RFID and WiFi, online location-based services and intelligent surveillance systems have attracted an increasing amount of attention [1–7]. Motivated by this, spatial-temporal queries for moving objects have been studied extensively in the fields of moving object databases (MOD) and geographical information systems (GIS) [8–11]. Many types of queries and corresponding efficient query algorithms have been proposed, such as range queries [12,13], *k*-nearest neighbor queries [14–16], reverse nearest neighbor queries [17], skyline queries [18], density queries [19,20] and visible nearest neighbor queries [21].

The widely-used and fundamental range queries for objects moving in Euclidean space or networks return a set of objects within a given area at a given query time [22–27]. This paper addresses the problem of processing continuous range queries for moving objects in networks. That is, querying objects and moving objects are both in high-speed motion and constrained by underlying networks. In contrast to snapshot range queries that are evaluated only once, the continuous range queries need

to maintain activity over a period of time and continuously retrieve query results. Many real-world applications depend substantially on these types of queries, such as fleet monitoring, vehicle rescue and traffic control and management [28,29]. For example, because of the long journey to the railway station, we first need to take a bus and then take a taxi. In this situation, we need to receive the continuous notification of when and where can we get a transfer to a nearby taxi within 500 meters from the running bus.

The continuous evaluation of range queries leads to increasing challenges to efficiency and server workloads due to ensuring the completeness and correctness of query results [19,30]. It is not easy to answer the continuous range queries for moving objects in networks over a period of time. There are two approaches for continuous range queries. The incremental approaches characterized by maximizing the reuse of previous results have better efficiency than other approaches that split the continuous range query into discrete snapshot range queries [22,29,31]. However, it has a strong dependence on the design of in-memory data structures to maintain the reusable candidates, such as an expansion tree and a safe region [13,25,32]. However, when the scale of networks is large, but the distribution of moving objects is relatively sparse, these data structures require more online re-computation to maintain the up-to-date query candidates, because the objects frequently enter and leave the regions of the expansion tree [16,17]. That invalidates these data structures and results in performance degradation. Thus, we need to develop novel data structures to maintain query candidates.

In this paper, we present a novel line graph-based continuous range (LGCR) query algorithm for moving objects in networks. First, the network is modeled as a line graph $G_{lg} = (V_{lg}, E_{lg})$. Segments are represented as graph nodes V_{lg}, and edges E_{lg} (i.e., spatial relationships) connect two adjacent road segments. Additionally, we also model moving objects as a graph structure G_{mo}. Based on the two relatively independent graph structures, a special edge $E_{mo \to lg}$ from a node of the object's graph G_{mo} to a node on the line graph G_{lg} represents the fact that an object is moving on a particular segment. An advantage of this design is that it provides more flexibility and control for the location update cost, because when a moving object turns into a new segment, this object only deletes the original edge and creates a new edge to reflect this change at the client side. Hence, this greatly reduces the location update cost and server workloads. Additionally, it is easier for the server to retrieve accurate moving objects using the special edge $E_{mo \to lg}$.

Additionally, we propose an innovative graph-based expansion tree (GET) structure that maintains the candidate moving objects. It creates a unique query node v_q for each query request and then creates a set of edges $E_{q \to lg}$ to represent candidate relationships between a query node v_q and line graph node v_{lg}. Hence, given that the update cost of GET depends on the cost of deleting or creating edges $E_{q \to lg}$, server workloads are reduced further. We implement the LGCR query algorithm and related data structures in the native graph database system Neo4J. We perform the experimental evaluation using a simulated dataset generated by GT-MobiSIMto demonstrate efficiency and performance compared with a classical continuous range query algorithm. The contributions of this paper lie in the following aspects:

- We develop a novel graph-based expansion tree (GET) based on the line graph model of networks. It supports offline pre-computation and could effectively reduce the online maintenance time of the traditional expansion tree in continuous queries.
- Based on GET, we propose a line graph-based continuous range (LGCR) query algorithm for moving objects in networks, including the algorithms for initialization, insertion, location update, filter and refinement.
- We conducted experiments to evaluate our proposed LGCR using real-world networks and simulated moving objects and compare with existing classical algorithms to verify its effectiveness.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the related data structures for continuous range queries in networks. Section 4

elaborates on the proposed LGCR query algorithm. Section 5 illustrates the experimental setting and results. Finally, Section 6 concludes the paper and proposes further developments.

2. Related Work

A straightforward approach to process continuous range queries is to split them into discrete snapshot range queries [8,16,33]. Wang et al. proposed the moving objects in road networks (MOVNet) algorithm to process continuous queries based on snapshot range queries for moving objects on road networks [34]. It introduced several significant features and a shortest distance-based tree (SD-tree), which was used to maintain network connectivity and network distance information to alleviate the effects of frequent object updates in continuous range queries. Kolahdouzan et al. developed an upper bound algorithm to improve the performance of continuous k-nearest neighbor queries by only performing snapshot queries at the location where they are required [16]. The UNICONS algorithm divided a query path into multiple subpaths based on the similarity principle and computed valid intervals. The query results for subpaths were combined to obtain the final results [8]. The advantage of this approaches is that the existing methods of snapshot range queries can be directly applied to continuous range queries without a drastic adjustment. However, the shortcoming is how to efficiently find the split points, which specify the location and time instant that keep the results of range queries the same. If a query algorithm is repeatedly evaluated at every time instant, performance obviously degrades. However, if a query algorithm is evaluated over long time intervals, query accuracy cannot be guaranteed.

Another widely-used strategy employs the incremental approaches adopted in our paper. The evaluation cost of continuous range queries could be reduced by reusing history results maintained by in-memory specific data structures. In particular, when updates from moving objects fall in this tree, they triggered a change of query results. In contrast, irrelevant updates were ignored [9,25]. Mouratidis et al. proposed an expansion tree structure in the incremental monitoring algorithm (IMA) and group monitoring algorithm (GMA) [32]. The algorithms start with the position of the querying object at network distance $q.kNN\_dist$, traverse around road segments and construct an expansion tree. Then, IMA/GMA monitor the changes of the query object and moving object's position, prune this expansion tree structure and reuse the candidate results to produce the query result of continuous queries. Similarly, the concept of the safe regions that calculate the closest moving objects to the border of the query was proposed to minimize the communication and computational cost of continuously monitoring a moving range query [35]. The concept of safe exits was proposed as the concise representation of the safe region that captures the border of the network-based safe region. Additionally, efficient algorithms for computing safe exits were developed [11,36]. Range Euclidean restriction (RER) and range network expansion (RNE) are well-known query processing algorithms based on this idea [37]. However, large-scale networks could decrease the performance of continuous queries based on the expansion tree because of high maintenance costs. Hence, the proposed GET in this paper expands and optimizes the original expansion tree structure. Simultaneously, it supports offline pre-computation to make our proposed have algorithm higher reliability for different sizes of networks.

3. Proposed Data Structures

In this section, we elaborate on related data structures in the LGCR query algorithm, which includes line graph, distance graph and GET.

Figure 1 illustrates an example of a city network G_{net} including 12 segments $(s_1, s_2, \ldots, s_{12})$ and 13 junctions $(j_1, j_2, \ldots, j_{13})$. Nine moving objects (green points) are moving along the network and sending location updates to a server. The query object q (yellow points) continually issues range queries over a period of time.

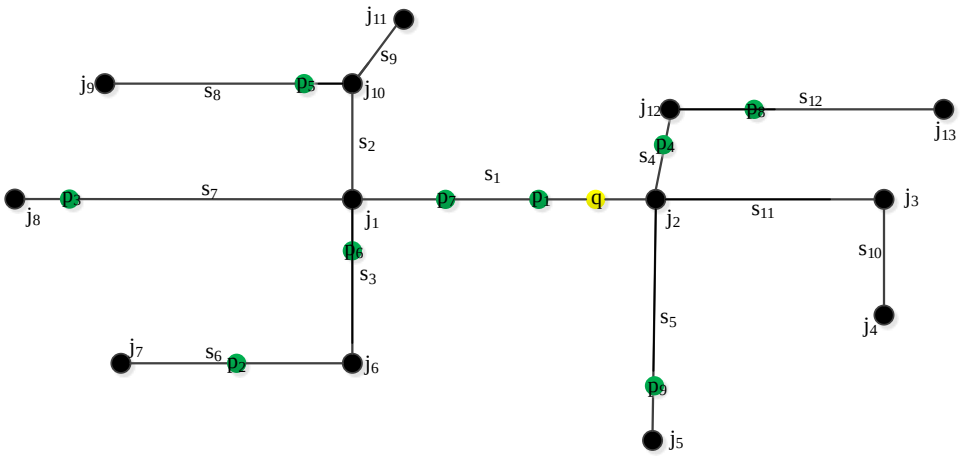


Figure 1. An example of a network.

(1) Basic types:

We present a set of basic types used for the definitions in the following parts. Three basic types are proposed for the following definitions:

$$int = Z \tag{1}$$

$$real = R \tag{2}$$

$$bool = \{true, false\} \tag{3}$$

Two time types are provided to represent the time:

$$instant = R \tag{4}$$

$$period = \{(s, e)|s, e \in instant\} \tag{5}$$

The geometry types are employed from OGC, which releases a series of specification about the geometry object model. The proposed LGCR involves three basic geometry types:

$$point = \{(lat, lon)|lat, lon \in real\} \tag{6}$$

$$line = \{< pt_1, pt_2, ..., pt_n > |n \in int, \forall i \in [1, n], pt_i \in point\} \tag{7}$$

$$polygon = \{< l_1, l_2, ..., l_n > |n \in int, \forall i \in [1, n], l_i \in line\} \tag{8}$$

(2) Line graph structure:

The network G_{net} includes a set of segments Seg. Let the domain of segments be defined as follows:

$$Seg = \{(sid, l, geo, start)|sid \in int, l \in real, geo \in line, start \in \{smaller, larger\}\} \tag{9}$$

with an identifier sid, length l and the flag $start$ that defines the orientation of a geometric line geo.

The line graph G_{lg} is another graph in which each vertex of G_{lg} represents a segment of G_{net}. The formal definition is given as follows:

$$G_{lg} = (V_{lg}, E_{lg}) \tag{10}$$

where:

$$V_{lg} = \{< seg_1, seg_2, ..., seg_n > |n \in int, \forall i \in [1, n], seg_i \in Seg\} \tag{11}$$

and:

$$E_{lg} = \{< grel_1, grel_2, ..., grel_n > |n \in int, \forall i \in [1, n], grel_i \in R_g\} \tag{12}$$

E_g represents a set of spatial relationships between two segments. We employ OGC-compliant nine-intersection model, which presents a comprehensive definition of topological relationships between spatial objects in two-dimensional space [38,39]. In this paper, topological relationships R_g are defined as follows:

$$R_g = \{(seg_i, seg_j, type) | seg_i, seg_j \in V_{lg}, type \in \{meet, equal\}\} \quad (13)$$

where the relationship *meet* denotes the case in which segments seg_i and seg_j are adjacent to each other and *equal* denotes the case in which two segments are the same.

Figure 2 illustrates an example of a line graph that corresponds to the network in Figure 1. The segment s_1 is modeled as a graph node. The connecting relationship between segments s_1 and s_2 is constructed as spatial relationships *meet*.

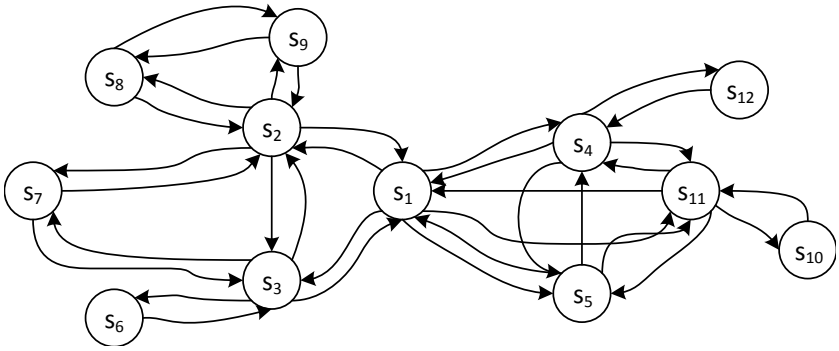


Figure 2. Line graph structure.

(3) Object graph structure:

A moving object is defined as follows:

$$Mo = \{(mid, name, param) | mid \in int, name \in string\} \quad (14)$$

where *mid* and *name* denote the identification and name of the object and *param* represents the set of other attributes of the object.

The object graph structure models moving objects and the social relations between them. It is defined as follows:

$$G_{mo} = (V_{mo}, E_{mo}) \quad (15)$$

where V_{mo} denotes moving objects and is defined as follows:

$$V_{mo} = \{< mo_1, mo_2, ..., mo_n > | n \in int, \forall i \in [1, n], mo_i \in Mo\} \quad (16)$$

and E_{mo} denotes the social relations between two moving objects and is defined as follows:

$$E_{mo} = \{< srel_1, srel_2, ..., srel_n > | n \in int, \forall i \in [1, n], srel_i \in R_{mo}\} \quad (17)$$

where:

$$R_{mo} = \{(mo_i, mo_j, type) | mo_i, mo_j \in V_{mo}, type \in string\} \quad (18)$$

and the *type* represents social relationships between moving objects.

(4) Network location

A moving object's network position $gloc$ is defined as follows:

$$gloc = \{(sid, d) | sid \in \text{int}, d \in real, 0 \leq d \leq l\} \tag{19}$$

where sid is an identifier of the segment and d represents a real number that provides a position on that segment.

An object's moving vector $mvector$ at time t is defined as follows:

$$mvector = \{(mid, gloc, v, t) | mid \in int, v \in real, t \in instant\} \tag{20}$$

where mid is an identifier of a moving object, $gloc$ denotes the objects' network position on a certain segment and v denotes the instantaneous velocity at time t.

(5) Bridgeable edges:

$E_{mo \to lg}$ connects a moving object and segment to represent the case in which a moving object moves on the segment and is defined as follows:

$$E_{mo \to lg} = \{< mlrel_1, mlrel_2, ..., mlrel_n > | n \in int, \forall i \in [1, n], mlrel_i \in R_{mo}^{lg}\} \tag{21}$$

where:

$$R_{mo}^{lg} = \{(mo_i, seg_j, type) | mo_i \in V_{mo}, seg_j \in V_{lg}, type \in string\} \tag{22}$$

Figure 3 illustrates an example of bridgeable edges, where a query object q travels from segment s_1 to segment s_2, as shown in Figure 3a. The edge (blue arrow line) from node s_1 to node q will be deleted, and a new edge (red arrow line) from node s_5 to node q will be created, as shown in Figure 3b.

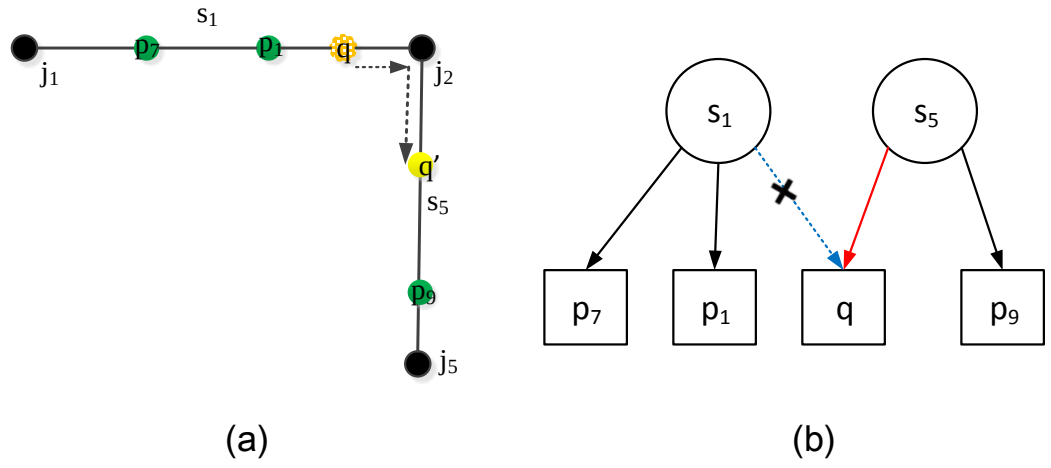


Figure 3. An example of bridgeable edges. (**a**) A query object in network; (**b**) The schematic of bridgeable edges

(6) Distance edges

The network distance calculation is the most time-consuming operation. Offline pre-computation network distance could effectively improve the algorithm's efficiency [16]; we create distance edges $E_{lg \to lg}$ to represent network distance. Then, we could use the convenient graph traversal operation to retrieve the distance value with lower query cost.

$E_{lg \to lg}$ represents a set of edges between any two line graph nodes. Its attributes store the network distance value between two segments and is defined as follows:

$$E_{lg \to lg} = \{ < llrel_1, llrel_2, ..., llrel_n > | n \in int, \forall i \in [1, n], llrel_i \in R_{lg}^{lg} \} \tag{23}$$

where:

$$R_{lg}^{lg} = \{ < seg_i, seg_j, d_{edge} > | seg_i, seg_j \in V_{lg}, d_{edge} \in real \} \tag{24}$$

We introduce the concept of edge distance d_{edge}, which is defined as the distance between two edges (i.e., road segments) according to the node types of the segments [40]. The edge distance can be classified into four types: SS, SE, ES, EE, where ES defines the distance from the end node E of $edge_i$ to the start node S of $edge_j$. Distance types SE, ES and EE are defined similarly.

As shown in Figure 4, the moving object qthat issues continuous range queries as it moves along segment s_2. $d_{es}(s_2, s_3)$ is defined as the distance from j_{2-e} to j_{3-s}, and $d_{es}(s_1, s_2)$ is defined as the distance from j_{1-e} to j_{2-s}.

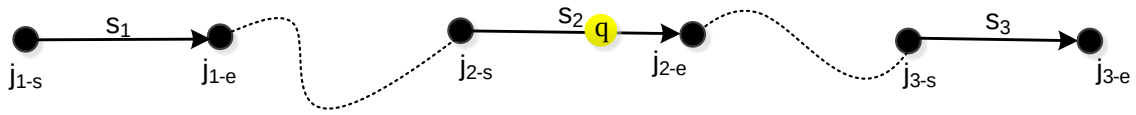


Figure 4. Distance edge.

(7) Graph-based expansion tree:

GET includes a set of segments and is defined as follows:

$$GET = \{(seg_q, (seg_1, ..., seg_n)) | n \in int, \forall i \in [1, n], seg_q, seg_i \in V_{lg}\}, \\ \forall i \in \{1, 2, ..., n\}, d_{edge}(seg_q, seg_i) < rg \tag{25}$$

where rg denotes the given distance parameters of continuous range queries and seg_q denotes the segment on which the query object moves. Clearly, segment seg_q and input parameters rg are critical for computing GET. Hence, for each segment in a network, we could precompute GET offline to improve the LGCR query algorithm's efficiency.

Figure 5 illustrates an example of GET. When the querying object q leaves the segment s_1, GET has to be recalculated to maintain up-to-date candidate results. Given that GET has already been precomputed, this significantly reduces the time required to process the LGCR query algorithm.

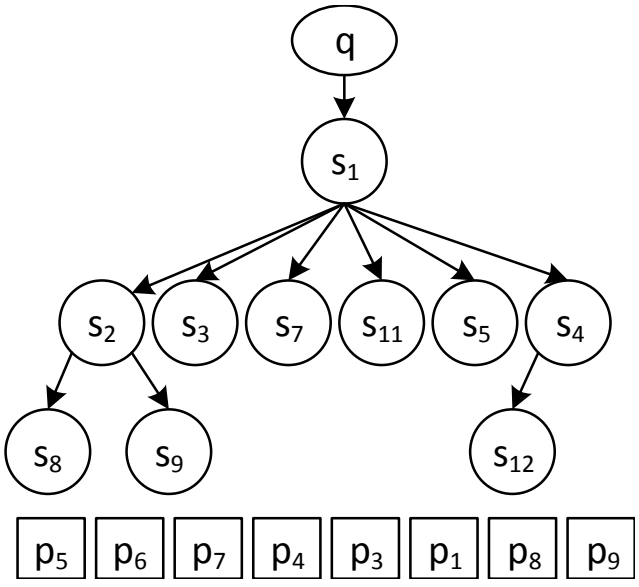


Figure 5. Graph-based expansion tree.

4. LGCR Query Algorithm

Based on the proposed data structures, the proposed LGCR query algorithm for moving objects in networks includes several algorithms for initialization, insertion, location update, filter and refinement, as shown in Figure 6.

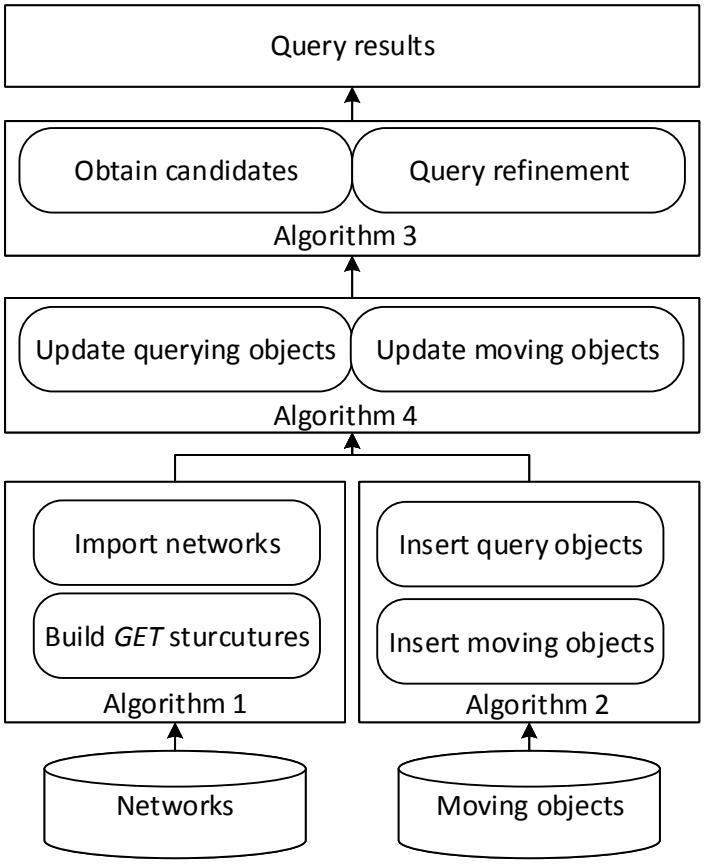


Figure 6. System architecture.

4.1. Algorithms

(1) Initialization:

The initialization of the LGCR query algorithm constructs the line graph structures and finishes the creation process of distance edges according to Algorithm 1. The input parameter of the algorithm is the network. To improve the efficiency of the LGCR query algorithm, we create an R-tree I_{rtree} to index the segments, as shown in Line 2, which traverses all of the segments and creates corresponding graph nodes on line graph G_{lg}, as shown in Lines 4–5. Then, the algorithm searches the adjacent segment nodes and creates connective edges to maintain complete topological relationships between the segments (Lines 8–10). Simultaneously, it constructs a minimum bounding rectangle (MBR) approximation of segments and links the leaf node of I_{rtree} to the corresponding segment nodes (Lines 6–7). Next, the algorithm traverses all segments to calculate the network edge distance between any two segments and constructs distance edges (Lines 13–20).

Finally, the algorithm traverses the segment node on the line graph and retrieves adjacent road segment nodes (Lines 22–23). Then, it obtains the value of the network edge distance d_{edge}. If the edge distance d_{edge} is less than the input continuous query range parameter rg, an edge is created and inserted into the line graph structure (Lines 24–27). Note that the initialization could only be executed once for the LGCR query algorithm.

Algorithm 1: Initialization algorithm.

Input: RN-network and rg-query range
Output: Line graph G_{lg}

```
1  Initialize line graph G_lg = Φ ;
2  I_rtree = Φ ;
3  for each segment seg in RN do
4      Create a node node_seg from seg in the line graph structure ;
5      G_lg.insertNode(node_seg) ;
6      Construct MBR from seg and insert it into I_rtree ;
7      CreateLink(index_leaf, node_seg) ;
8      for each node_seg^adj in the located adjacent nodes of node_seg do
9          Create a connective edge grel = (node_seg, node_seg^adj) ;
10         G_lg.insertNode(grel) ;
11     end
12     for each segment node node_seg in line graph G_lg do
13         for each segment node node_seg^other in line graph G_lg do
14             if node_seg ≠ node_seg^other then
15                 Calculate the edge distance d_edge between two segments ;
16                 Create a distance edge llrel = (node_seg, node_seg^other) with attribute d_edge ;
17                 G_lg.insertEdge(llrel) ;
18             end
19         end
20     end
21     for each segment node node_seg in line graph G_lg do
22         for each node_seg^adj in located adjacent nodes of node_seg do
23             if distance edge d_edge < rg then
24                 Create an edge e_get = (node_seg, node_seg^adj) of GET ;
25                 G_lg.insertEdge(e_get) ;
26             end
27         end
28     end
29 end
30 return G_lg;
```

(2) Insertion of moving objects and query objects:

As illustrated in Algorithm 2, the insertion of the moving object constructs a node. Then, the algorithm locates corresponding road segments on which this object is moving and builds bridgeable edges $E_{mo \to lg}$ (Lines 3–7).

Algorithm 2: Insertion of moving objects and query objects.

Input: *MO*-a new moving object
Output: Object graph G_{mo}

```
1 Create a node node_mo from mo in the object graph G_mo ;
2 G_mo.insertNode(node_mo) ;
3 Search I_rtree on the current location of mo and locate leaf node entries ;
4 for each nodept in located I_rtree leaf node entries do
5 |   Locate node_seg using the pointer nodept ;
6 |   Create an edge mlrel = (node_mo, node_seg) ;
7 |   G_mo.insertEdge(mlrel) ;
8 end
9 return G_mo;
```

(3) Filter and refinement step for the LGCR query algorithms:

Algorithm 3 illustrates the filter and refinement step of the LGCR query algorithm. The filter step traverses GET to locate all of the candidate moving objects using bridgeable edges $E_{mo \to lg}$, as shown in Line 1. The refinement step is performed periodically. The algorithm traverses the candidate results and determines whether the moving object's current time and network distance to the query object satisfy the given continuous range query condition (Lines 3–8). If the lifecycle of the LGCR query algorithm contains the object's current time and the object's location intersects the query range, then the candidate object is moved into the set of incremental query results (Line 6).

Algorithm 3: Filter and refinement step.

Output: *AnswerSet* (query results, a set of objects that meet the range query condition)

```
1 Get candidate results filterset from expansion tree GET ;
2 for each moving object node_mo^can in filterset do
3 |   if node_qr.st < node_mo^can.curtime and node_qr.et > node_mo^can.curtime then
4 |   |   Calculate the network distance dist between node_mo^q and node_mo^can ;
5 |   |   if dist ≤ node_qr.rg then
6 |   |   |   AnswerSet.addResults(node_mo^can) ;
7 |   |   end
8 |   end
9 end
```

(4) Location update of moving objects:

There are three types of location update strategies for moving objects: speed-threshold-triggered location update (STTLU), distance-threshold-triggered location update (DTTLU) and ID-triggered location update (IDTLU) [41]. Given that the location update will be triggered by the moving object turning into a new road segment, the IDTLU strategy is more appropriate for the LGCR query algorithm. Algorithm 4 manages those location updates. First, it locates the object $node_{mo}$ that needs to be updated according to the identifier *mid*. Simultaneously, the old road segment $node_{lg}^{old}$ could be easily retrieved using the bridgeable edges $E_{mo \to lg}$ (Lines 1–2). Then, Lines 3–4 update the object $node_{mo}$ value with a new location and time from the location updates. Additionally, it creates a new bridgeable edge to a new segment. Next, when the object belongs to the query object, GET needs to be updated, as shown in Line 7. Otherwise, the filter and refinement step needs to be performed (8–12).

Algorithm 4: Location update of a moving object or query object.

Input: new location and time of a moving object mo

Find object node $node_{mo}$;
Get the old road segment node $node_{lg}^{old}$ using edge $mlrel$ of $node_{mo}$;
Update $node_{mo}$ with the new location and time ;
Get the new road segment node $node_{seg}^{new}$ using edge $mrel$ of $node_{mo}$;
if $node_{seg}^{new}.sid! = node_{seg}^{old}.sid$ **then**
    if $node_{mo}$ *is a query object* **then**
        Get the new GET structure according to the road segment $node_{seg}^{new}$;
    end
    else if $node_{mo}$ *is a moving object* **then**
        if $node_{seg}^{new}$ *has an edge* e_q *linked to the query object* **then**
            $FilterRefinementStep()$ (Algorithm 3) ;
        end
    end
end

4.2. Analysis of the Algorithm's Complexity

Assume that there are E edges on networks, M moving objects and Q query objects. The time complexity of the LGCR query algorithm could be defined as follows:

$$T = T_{filter} + T_{up} + T_{ref} \tag{26}$$

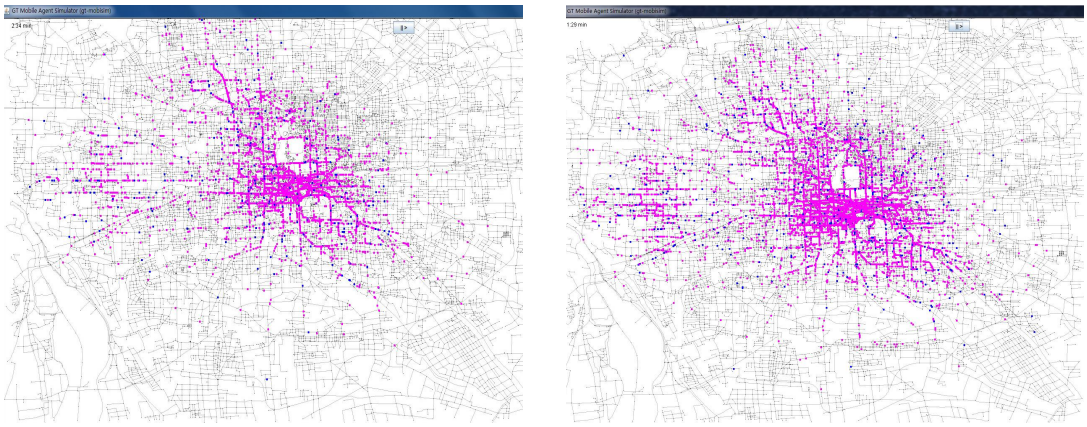
where T_{filter} denotes the CPU time required for the calculation of query candidates obtained by GET, T_{up} denotes the CPU time required for location updates and T_{ref} denotes the CPU time required for the refinement step. The time required to retrieve query candidates includes the GET traversal time. Then, the complexity T_{filter} of this algorithm is defined as $O(Qn)$. The location update of objects triggers an edge operation, and the time complexity of T_{up} is $O(Mn)$. The time required to calculate the network distance depends on the graph traversal time because of the distance edges. Then, the time complexity T_{ref} depends on the size of the candidate results and is defined as $O(\frac{Q}{M}n)$.

5. Experiments

5.1. Experimental Settings

We conducted experiments on a standard personal computer with an Intel i7-4790 CPU, 8G RAM and a 1-TB mechanical hard disk. We used the generator GT-MobiSIM [42] driven by an XML configuration file to simulate moving objects, and query objects were randomly selected to issue continuous range query requests. In order to test the performance and efficiency of our proposed LGCR, we prepared two experiments. The first experiment simulated 5000 moving objects in a network and randomly selected 500 query objects to issue continuous range query requests with different range parameters, as shown in Figure 7a. The second experiment simulated 10,000 moving objects and randomly selected 1000 query objects, as shown in Figure 7b. As illustrated in the figure, the red points were moving objects, and the blue points were querying objects. The input employed real network datasets from Beijing, with 26,220 segments and 18,856 intersections. The generator included various mobility models in networks, such as random waypoint and random trip. Additionally, there were three ways to represent continuous-time traces, including the location step-function, velocity set-function and acceleration step-function. It also included various parameter distributions for moving objects, such as the Gaussian distribution and normal distribution. In experiments, we set the parameter distribution to the Gaussian distribution, and each moving object followed the shortest path

to the final destination generated from a predefined random waypoint mobility model. The related data structures were implementation using the native graph database Neo4J 1.9.7. The proposed LGCR query algorithm was implemented using Java as the main programming language and Eclipse as the development environment.



(a) **(b)**

Figure 7. GT-MobiSIM. (**a**) The simultion with 5000 moving objects and 500 query objects; (**b**) The simultion with 10,000 moving objects and 1000 query objects.

For comparison, we used the classical algorithm proposed by Stojanovic et al., known as StojanovicCR [13] for a continuous range query over moving objects in networks with high efficiency and effectiveness, which satisfied real-world, location-based services and applications. This algorithm generated a series of in-memory data structures, segment R*-tree (SR*-tree), continuous query table (CQT), network connectivity table (NCT) and mobile object table (MOT), to support the evaluation of the continuous range query. In particular, the SR*-tree extended R*-tree to store MBR information of road segments. The leaf node entry of SR*-tree was represented as $(segid, mbr, olist, qlist)$, where mbr was the minimum bounding rectangle, $olist$ and $qlist$ denoted the list of objects and querying objects moving along the road segment with the identifier $segid$. The NCT table stored connectivity information of the road segments. An NCT table entry was represented as $(segid, rtEntry, segLength, startCon, endCon)$, where $segid$ was the identifier of road segment, $rtEntry$ denoted the pointer to SR*-tree, $segLength$ represented the length of road segment an the elements $startCon$ and $endCon$ denoted the lists of adjacent road segments. The CQT table stored information of continuous range queries. A CQT entry was represented as $(qid, oid, range, validPeriod, answerSet)$, where qid was the identifier of continuous range queries, oid denoted the querying object, $range$ defined the range parameters, $answerSet$ denoted the initial query answer and $validPeriod$ represented the valid period of the query. The MOT table stored the information of moving objects. A MOT entry was represented as $(moid, loc, time, speed, querySet)$, where $moid$ was the identifier of moving objects, loc was the current location at the time $time$and $speed$ denoted the current velocity. The $querySet$ represented the list of queries in which this object participates. The innovative StojanovicCR introduced a pre-refinement step to further refine the moving objects by the filter step in a traditional query processing strategy. First, according to the query condition, the filter-step selected the candidate moving objects using the SR*-tree index. The pre-refinement step was to further refine the moving objects obtained by the filter step. The refinement step was preformed periodically by processing the candidate objects generated by the pre-refinement step and generated the query answer.

5.2. Experimental Results

The initialization of LGCR took about 983 seconds in the experiment. It mainly included the three following parts: importing networks, building the proposed data structures and constructing

the GET structure. GET is the key to improving the performance of the LGCR query algorithm in an incremental manner. According to the query range, queries are classified into different groups. We calculate the average calculation time of the query candidate for each group with the same query range. Figure 8 illustrates the time taken to retrieve candidate objects using GET. The results show that retrieving candidate objects required more CPU time as the query range increased. Given that the basic operation is graph traversal in GET, the large query range requires more graph traversal operations, which results in more time consumed.

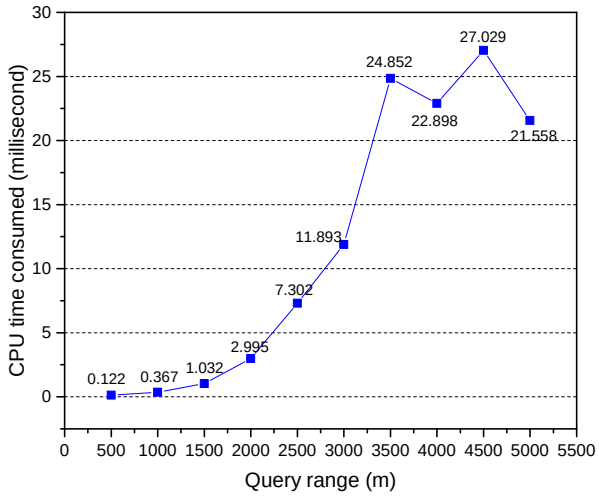


Figure 8. The CPU time needed for the calculation of the query candidate.

Figure 9 illustrates that the average number of moving objects in the query results (orange bar) is very close to the average number of moving objects in the candidate results (blue bar). The average utilization ratio of candidate objects retrieved by GET increased to 81.2%, that is GET effectively maintained the reusable candidate moving object for incremental continuous range queries for the LGCR query algorithm. The basic unit of GET is a road segment. Hence, the length of a road segment is related to the number of candidates. A longer road segment led to the ratio between the query results and candidate results decreasing. The average length of road segments in Beijing in the experiments was 202.25 meters, which is a relatively short distance; hence, GET maintains more accurate candidate sets.

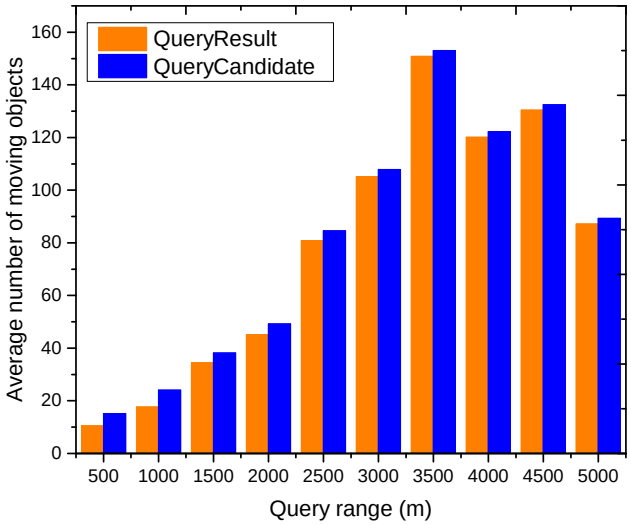


Figure 9. The average number of moving objects obtained in query results and the query candidate.

Figure 10a shows the average CPU time per query in the experiment with 5000 moving objects and 500 query objects. Figure 10b illustrates the experimental results with 10,000 moving objects and 1000 query objects. The experimental results show that the LGCR query algorithm provided better performance and stability for the evaluation of continuous queries than StojanovicCR. The average CPU time per query for both approaches generally increased as the query range increased. StojanovicCR required more time to update in-memory data structures, such as SR*-tree, MOT and CQT. The LGCR query algorithm also triggered longer execution times for query candidates for continuous range queries. However, the basic graph operators, such as creating or deleting edges and graph traversal, were less time consuming. Hence, the LGCR query algorithm was more stable and efficient. In addition, the proposed distance edges and GET support offline pre-computation and could further improve the LGCR's efficiency.

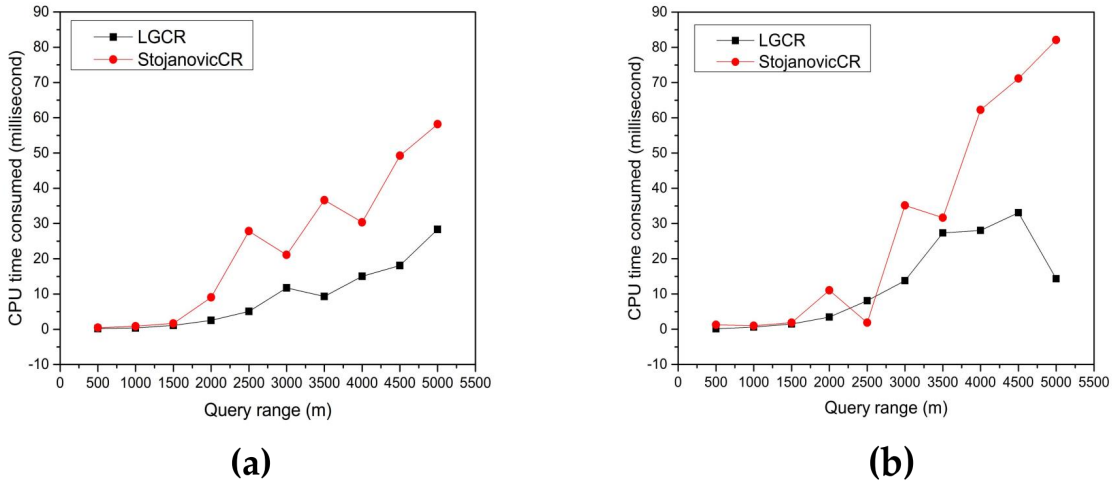


Figure 10. Comparison of Stojanovic continuous range (StojanovicCR) and line graph-based continuous range (LGCR). (**a**) The result of the experiment with 5000 moving objects and 500 query objects; (**b**) The result of the experiment with 10,000 moving objects and 1000 query objects.

5.3. Discussion

The proposed LGCR query algorithm was efficient for continuous range queries over moving objects in networks. However, there are still limitations worthy of attention.

(1) For the LGCR query algorithm, special data structures, including bridgeable edges, distance edges and the GET and offline precomputation steps, significantly improved efficiency. However, an oversized network scale made these preprocessing steps very time consuming and required massive storage space. In addition, our proposed LGCR could be easily extended to support geo-social queries because that the ObjectGraph structure has the capability of modeling social relationships between moving objects, such as colleagues, friendships, followership, interest groups or fan relationships.

(2) As the key task of the LGCR query algorithm, the basic unit of GET contained a set of segments. However, those segments had no obvious physical meaning; they could not adapt well to conceptual entities in real-world complex road network environments [2,15]. The concept of routes corresponding to highways or expressways in the real world could be introduced to model road networks. Hence, a composite GET that contains both road segments and routes could further improve the flexibility of the LGCR query algorithm.

(3) We assumed that the network structure changed little and that updates mainly considered the location updates of moving objects. However, in a real-world scenario, the weight of a segment always changes with traffic information, and new road segments might be inserted. These influencers would affect the query results of continuous range queries. Hence, more types of updates should be considered to make the LGCR query algorithm more extensive.

6. Conclusions

Motivated by a substantial need for location-based services and applications for spatial-temporal queries, we presented a novel LGCR query algorithm that depended on a line graph structure and GET structure that could solve the problem of the efficient processing of continuous range queries over massive moving objects in networks. The algorithm can be applied directly to real-world location-based services and applications.

Future work needs to implement parallel processing of continuous range queries based on the LGCR query algorithm to further improve performance. The benefits of the proposed data structure include the line graph mode and GET structure; the proposed LGCR query algorithm is easy to deploy in distributed computing environments with the help of a large-scale graph commutating processing framework, such as Pregel and bulk synchronous parallel (BSP).

Acknowledgments: This research was supported by the State's Key Project of Research and Development Plan (Grant No. 2016YFB0502104) and the National Natural Science Foundation of China (Grant No. 41401460, 41571431, 41421001). Additionally, we would also like to thank the anonymous referees for their helpful comments and suggestions.

Author Contributions: Hengcai Zhang and Feng Lu provided the core idea for this study. Hengcai Zhang implemented the LGCR query algorithm and carried out the experimental validation. Hengcai Zhang and Feng Lu wrote the main manuscript. Jie Chen made the important comments and suggestions for this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wolfson, O.; Xu, B.; Chamberlain, S.; Jiang, L. Moving objects databases: Issues and solutions. In Proceedings of the 10th International Conference on Scientific and Statistical Database Management, Washington, DC, USA, 1–3 July 1998.
2. Güting, R.H.; Ding, Z. Modeling and querying moving objects in networks. *Int. J. Very Large Data Bases* **2006**, *15*, 165–190.
3. Ciuonzo, D.; Buonanno, A.; D'Urso, M.; Palmieri, F.A. Distributed classification of multiple moving targets with binary wireless sensor networks. In Proceedings of the 2011 Proceedings of the 14th International Conference on Information Fusion (FUSION), Chicago, IL, USA, 5–8 July 2011.
4. Buonanno, A.; D'Urso, M.; Prisco, G.; Felaco, M.; Meliadò, E.; Mattei, M.; Palmieri, F.; Ciuonzo, D. Mobil sensor networks based on autonomous platforms for homeland security. In Proceedings of the 2012 Tyrrhenian Workshop on Advances in Radar and Remote Sensing (TyWRRS), Naples, Italy, 12–14 September 2012.
5. Parent, C.; Spaccapietra, S.; Renso, C.; Andrienko, G.; Andrienko, N.; Bogorny, V.; Damiani, M.L.; Gkoulalas-Divanis, A.; Macedo, J.; Pelekis, N. Semantic trajectories modeling and analysis. *ACM Comput. Surv.* **2013**, *45*, 1–42.
6. Shekhar, S.; Jiang, Z.; Ali, R.Y.; Eftelioglu, E.; Tang, X.; Gunturi, V.; Zhou, X. Spatiotemporal Data Mining: A Computational Perspective. *ISPRS Int. J. Geo-Inf.* **2015**, *4*, 2306–2338.
7. Zheng, Y. Trajectory data mining: An overview. *ACM Trans. Intell. Syst. Technol.* **2015**, *6*, 1–29.
8. Cho, H.-J.; Ryu, K.; Chung, T.-S. An efficient algorithm for computing safe exit points of moving range queries in directed road networks. *Inf. Syst.* **2014**, *41*, 1–19.
9. Xuan, K.; Zhao, G.; Taniar, D.; Rahayu, W.; Safar, M.; Srinivasan, B. Voronoi-based range and continuous range query processing in mobile databases. *J. Comput. Syst. Sci.* **2011**, *77*, 637–651.
10. Zhu, T.; Wang, C.; Lv, W.; Huang, J. Continuous range monitoring of moving objects in road networks. In Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications (ISDA), Cairo, Egypt, 29 November–1 December 2010.
11. Yung, D.; Yiu, M.L.; Lo, E. A safe-exit approach for efficient network-based moving range queries. *Data Knowl. Eng.* **2012**, *72*, 126–147.
12. Zheng, K.; Trajcevski, G.; Zhou, X.; Scheuermann, P. Probabilistic range queries for uncertain trajectories on road networks. In Proceedings of the Proceedings of the 14th International Conference on Extending Database Technology, Uppsala, Sweden, 21–24 March 2011.

13. Stojanovic, D.; Papadopoulos, A.N.; Predic, B.; Djordjevic-Kajan, S.; Nanopoulos, A. Continuous range monitoring of mobile objects in road networks. *Data Knowl. Eng.* **2008**, *64*, 77–100.
14. Huang, Y.K.; Chen, Z.W.; Lee, C. Continuous k-nearest neighbor query over moving objects in road networks. *Adv. Data Web Manag.* **2009**, *5446*, 27–38.
15. Jensen, C.S.; Kolářvr, J.; Pedersen, T.B.; Timko, I. Nearest neighbor queries in road networks. In Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, New Orleans, LA, USA, 3–8 November 2003.
16. Kolahdouzan, M.R.; Shahabi, C. Continuous k-nearest neighbor queries in spatial network databases. In Proceedings of the STDBM'04, Toronto, ON, Canada, 30 August 2004.
17. Safar, M.; Ibrahimi, D.; Taniar, D. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Syst.* **2009**, *15*, 295–308.
18. Safar, M.; El-Amin, D.; Taniar, D. Optimized skyline queries on road networks using nearest neighbors. *Pers. Ubiquitous Comput.* **2011**, *15*, 845–856.
19. Hao, X.; Meng, X.; Xu, J. Continuous density queries for moving objects. In Procedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, Vancouver, BC, Canada, 13 June 2008.
20. Jensen, C.S.; Lin, D.; Ooi, B.C.; Zhang, R. Effective Density Queries on Continuously Moving Objects. In Procedings of the International Conference on Data Engineering, Atlanta, GA, USA, 3–8 April 2006.
21. Gao, Y.; Zheng, B.; Chen, G.; Li, Q.; Guo, X. Continuous visible nearest neighbor query processing in spatial databases. *VLDB J.* **2011**, *20*, 371–396.
22. Alamri, S.; Taniar, D.; Safar, M. A taxonomy for moving object queries in spatial databases. *Future Gener. Comput. Syst.* **2014**, *37*, 232–242.
23. Gaede, V.; Günther, O. Multidimensional access methods. *ACM Comput. Surv.* **1998**, *30*, 170–231.
24. Guo, X.; Zheng, B.; Ishikawa, Y.; Gao, Y. Direction-based surrounder queries for mobile recommendations. *VLDB J.* **2011**, *20*, 743–766.
25. Jung, H.; Kim, Y.S.; Chung, Y.D. QR-tree: An efficient and scalable method for evaluation of continuous range queries. *Inf. Sci.* **2014**, *274*, 156–176.
26. Tao, Y.; Xiao, X.; Cheng, R. Range search on multidimensional uncertain data. *ACM Trans. Database Syst.* **2007**, *32*, 1–15.
27. Xu, J.; Güting, R.H.; Zheng, Y. The TM-RTree: An index on generic moving objects for range queries. *GeoInformatica* **2015**, *19*, 487–524.
28. Sowell, B.; Salles, M.V.; Cao, T.; Demers, A.; Gehrke, J. An experimental analysis of iterated spatial joins in main memory. *Proc. VLDB Endow.* **2013**, *6*, 1882–1893.
29. Taniar, D.; Rahayu, W. A taxonomy for region queries in spatial databases. *J. Comput. Syst. Sci.* **2014**, *81*, 1508–1531.
30. Huang, Y.K.; Lin, L.F.; Chung, Y.C.; Su, I.F. Continuous min-max distance bounded query in road networks. In *Web Technologies and Applications*; Springer: Berlin, Germany, 2012; pp. 423–434.
31. Long, J.A.; Nelson, T.A. A review of quantitative methods for movement data. *Int. J. Geogr. Inf. Sci.* **2013**, *27*, 292–318.
32. Mouratidis, K.; Yiu, M.L.; Papadias, D.; Mamoulis, N. Continuous nearest neighbor monitoring in road networks. In Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, 12–15 September 2006.
33. Huang, Y.-K.; Lin, L.-F. Efficient processing of continuous min–max distance bounded query with updates in road networks. *Inf. Sci.* **2014**, *278*, 187–205.
34. Wang, H.; Zimmermann, R. Processing of continuous location-based range queries on moving objects in road networks. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1065–1078.
35. AL-Khalidi, H.; Taniar, D.; Betts, J.; Alamri, S. Dynamic safe regions for moving range queries in mobile navigation. *Int. J. Ad Hoc Ubiquitous Comput.* **2014**, *16*, 250–259.
36. Cheema, M.A.; Brankovic, L.; Lin, X.; Zhang, W.; Wang, W. Continuous monitoring of distance-based range queries. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1182–1199.
37. Papadias, D.; Zhang, J.; Mamoulis, N.; Tao, Y. Query Processing in Spatial Network Databases. In Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany, 9–12 September 2003.
38. Egenhofer, M.J.; Franzosa, R.D. Point-set topological spatial relations. *Int. J. Geogr. Inf. Syst.* **1991**, *5*, 161–174.

39. Egenhofer, M.J.; Herring, J. A Mathematical Framework for the Definition of Topological Relationships. In Proceedings of the Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland, 23–27 July 1990.
40. Liu, F.; Do, T.; Hua, K. Dynamic range query in spatial network environments. In *Database and Expert Systems Applications*; Springer: Berlin, Germany, 2006; pp. 254–265.
41. Ding, Z.; Guting, R.H. Managing moving objects on dynamic transportation networks. In Proceedings of the 16th International Conference on Scientific and Statistical Database Management, Santorini Island, Greece, 21–23 June 2004.
42. GT-Mobisim. 2015. Available online: https://github.com/Sdcxv/gt-mobisim (accessed on 14 December 2016).